ANL/NSE-21/82

Machine Learning Assisted Safety Modeling and Analysis of Advanced Reactors

Nuclear Science and Engineering Division
Mathematics and Computer Science Division
Computational Science Division

ANL/NSE-21/82

Machine Learning Assisted Safety Modeling and Analysis of Advanced Reactors

prepared by

Yang Liu[1], Rui Hu[1], Dezhi Dai[1], Prasanna Balaprakash[2], and Aleksandr Obabko[3]

[1]Nuclear Science and Engineering Division, Argonne National Laboratory
[2]Mathematics and Computer Science Division, Argonne National Laboratory
[3]Computational Science Division, Argonne National Laboratory

December 20, 2021

Executive Summary

With the advances in computational power and numerical methods, analysts can now rely on first-principle simulations to predict ultra-fine details in a variety of applications. Advances in machine learning (ML) have produced algorithms that can now learn high-level abstractions via hierarchical models. This project aims to leverage advances in ML techniques and the available high-resolution simulation data to develop a novel modeling and simulation (M&S) methodology for reactor safety analysis. While application-agnostic ML techniques are available, complex physics constraints need to be incorporated into ML techniques to build ML-based closures for computationally efficient predictive simulations.

This project intends to develop a physics-guided data-driven multi-scale methodology for M&S of advanced reactors. The project focuses on thermal fluid (T/F) phenomena, which play major roles in advanced reactor safety. Specifically, we propose a data-driven coarse-mesh turbulence model based on local flow features for the transient analysis of thermal mixing and stratification in a sodium-cooled fast reactor (SFR). The model has a coarse-mesh setup to ensure computational efficiency, while it is trained by fine-mesh computational fluid dynamics (CFD) data with Reynolds-averaged Navier-Stokes (RANS) turbulence model to ensure accuracy. Three different neural networks are developed and tested for loss-of-flow transients in the hot pool of SFR, i.e. the densely connected convolutional neural network (DCNN), long-short-term-memory network based on proper orthogonal decomposition (POD-LSTM), and the DCNN informed by LSTM (DCNN-LSTM). The performances of these three neural networks are evaluated based on baseline models. The DCNN-LSTM model has been chosen for further hyperparameter optimization.

Furthermore, based on a simplified two-dimensional case, uncertainty quantification (UQ) of the developed ML-based closure are investigated with three methods, i.e. Monte Carlo dropout, deep ensemble, and Bayesian neural network. The developed ML-based turbulent viscosity closure relation based on deep ensemble is then integrated into the system analysis module SAM and serves as a term in the conservation equations. Such a SAM-ML based procedure guarantees that the obtained results are consistent with the physical constraints of the thermal-fluid system. The SAM-ML simulation on the same loss-of-flow transient showed comparable accuracy with the CFD simulation but with a much coarser mesh setup.

Last but not least, the ML-based closure improvement with the support of higher-fidelity data from large eddy simulation (LES) is discussed. As a first step towards this direction, a baseline LES simulation is performed to obtain comparable data with RANS results. Based on the early results, future investigation on further improving the ML-based closure is discussed.

We believe the developed approach that combines scientific machine learning with nuclear system analysis code can benefit the advanced reactor community as more accurate safety analyses will better characterize reactor safety margins and reduce licensing efforts.

Contents

List of Figures

1 Introduction

1.1 Research goal

Advanced reactors are expected to fulfill a key role in next-generation nuclear power plants because of their increased safety and economic performance. Leveraging the full potential of these new reactor technologies requires modern analysi]s tools to ensure the safety of the new designs. As an example, the System Analysis Module (SAM), a modern system-level code, is currently under active development at Argonne National Laboratory for advanced reactor design and safety analysis [1].

Although the main objective of a system-level code is to conduct whole-plant transient analysis, three-dimensional (3-D) simulation capability is still desirable to tackle complex thermal-fluid (T-F) phenomena in advanced reactors. A typical example is the mixing and thermal stratification phenomena in large pools or enclosures. Such phenomena can be observed in various reactor systems, including the cold- and hot-pool mixing in pool-type sodium-cooled fast reactors, reactor cavity cooling system behavior in high-temperature gas-cooled reactors, passive containment cooling in advanced light-water reactors, and thermal stratification in boiling-water reactor suppression pools.

It is very important to accurately predict pool temperature and density distributions for both design optimizations and safety analyses of these reactor systems. However, the individual transport mechanisms governing mixing are characterized by time and length scales that can differ by orders of magnitude. Large volumes and complex interactions of different flow and thermal structures make the analysis of mixing in a large enclosure a very challenging task. Current major reactor system analysis codes have either no models or only coarse one-dimensional (1-D) models for thermal mixing and stratification in large enclosures. The lack of general thermal mixing and stratification models in those codes severely limits their application and accuracy for safety analysis, especially for reactors relying on natural circulation for long-term cooling.

For this reason, a 3-D module is currently being developed in SAM to accurately model complex T-F phenomena while maintaining the computational efficiency of system code [2]. This module adopts a coarse-mesh setup to be consistent with the one-dimensional system modeling framework, which ensures computational efficiency. A major challenge for the coarse-mesh 3-D module in SAM is to accurately capture turbulence [3] in reactor transients. A widely used approach for turbulence prediction in such engineering applications is the two-equation turbulence model based on the Reynolds-averaged Navier-Stokes (RANS) equations. However, this approach still requires a fine-mesh setup, so it is incompatible with the coarse-mesh 3-D module in system code such as SAM.

Data-driven approaches enabled by machine learning (ML) have led to significant progress in many nuclear engineering applications. Among various ML methods, deep neural networks (DNNs) provide us with an especially promising technical approach to developing a data-driven coarse-mesh turbulence model. DNNs have demonstrated promising results in achieving approximate solutions of partial differential equations (PDEs), with two advantages: First, DNNs are capable of modeling nonlinear relationships between inputs and outputs, as proven by the universal approximation theorems [4]. Second, once the DNN is trained, its forward evaluation is very fast, making it computationally preferable to introducing additional PDEs to the system code for turbulence prediction. These advantages make the DNN-based data-driven turbulence model possible for nuclear-system code. Leveraging DNNs, we can learn from the high-resolution CFD data to develop coarse mesh model that is able to retain the accuracy of the CFD results.

In this report, we present a comprehensive framework that integrates ML-based closures with nuclear system code for improved modeling and simulation (M&S) of advanced reactor transients that involve complex T-F phenomena, as illustrated in Figure 1.1. Specifically, we leverage DNN to develop a data-driven coarse-mesh turbulence model for modeling reactor transients that involve thermal mixing/stratification in large open volumes. The DNN model is trained with fine-mesh CFD results to ensure accuracy, while it has a coarse-mesh setup so its computational efficiency is similar to that of system code. We adopted the convolutional recurrent neural network architecture to capture the spatial-temporal information from CFD simulation results during reactor transients. We performed a case study on a simplified Sodium Fast Reactor (SFR) hot pool with transient simulations to demonstrate the applicability of the developed DNN model. Furthermore, in a simplified 2-D case, we investigated the uncertainty quantification (UQ) of the obtained ML-based closure with three methods, i.e. Monte Carlo dropout, deep ensemble, and Bayesian neural network. We then incorporate the UQ capability in the ML-based closure with deep ensemble. We then integrate the obtained deep ensemble based ML closure into SAM which serves as a term in the conservation equations. Such a SAM-ML based procedure guarantees that the obtained results are consistent with the physical constraints of the thermal-fluid system. Last but not least, the ML-based closure improvement with the support of higher-fidelity data from large eddy simulation (LES) is discussed. As a first step towards this direction, a baseline LES simulation is performed to obtain comparable data with RANS results. Based on the early results, future investigation on further improve the ML-based closure is discussed.

The rest of this paper is organized as follows: the rest of Section 1 discusses the problem; 2 introduces the three proposed neural network architectures; Section 3 presents a complete case study for developing the ML-based closure; Section 4 investigates the uncertainty quantification of ML-based closure and its implementation into SAM; Section 5 discusses improving the ML-based closure with higher-fidelity data; and Section 6 provides summary remarks and future investigation direction.

1.2 Coarse mesh multi-dimensional module in SAM

SAM's multi-D module is based on three conservation equations including mass, momentum, and energy:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot (\rho \boldsymbol{V}) = 0 \tag{1}$$

$$\rho \frac{\partial \boldsymbol{V}}{\partial t} + \rho \left(\boldsymbol{V} \cdot \nabla \right) \boldsymbol{V} + \nabla \mathrm{p} - \rho \boldsymbol{g} - \nabla \cdot \boldsymbol{\tau} = 0 \tag{2}$$

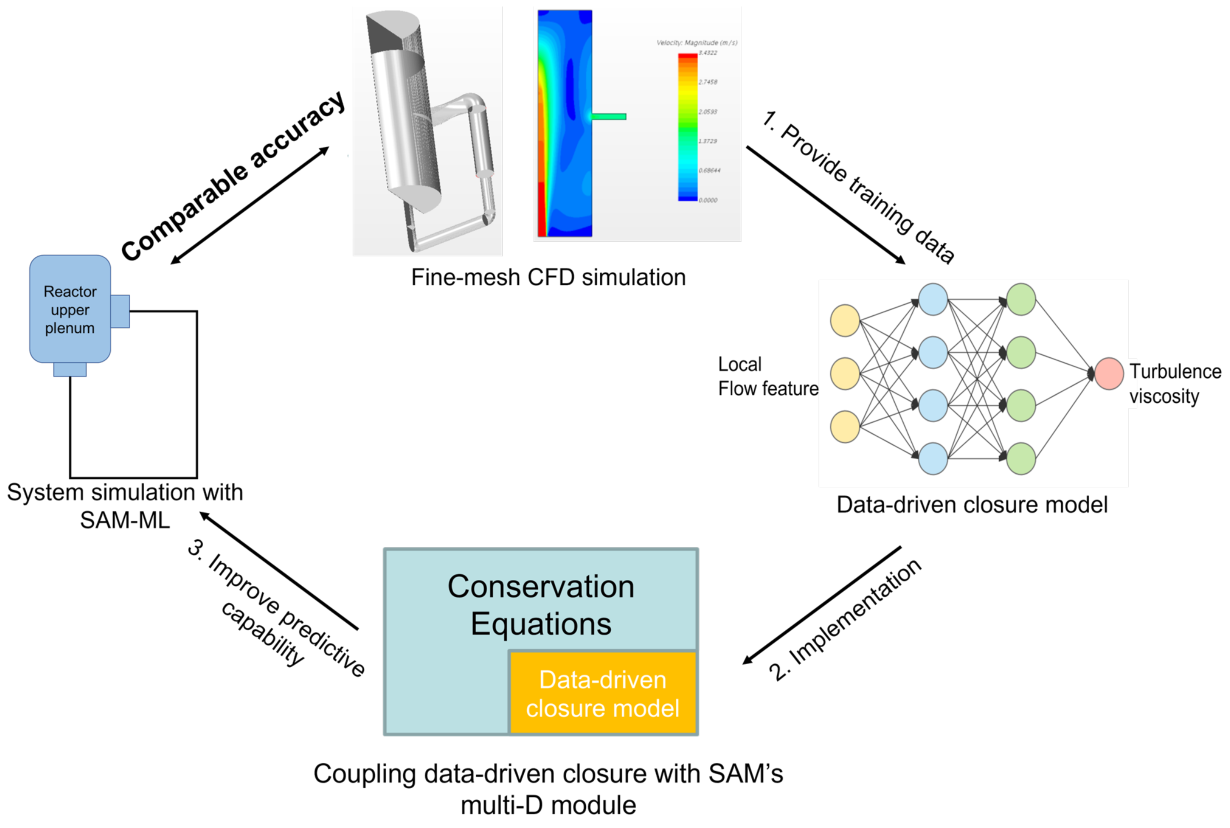

Comparable accuracy
Fine-mesh CFD simulation
1. Provide training data
Reactor upper plenum
Local Flow feature
Turbulence viscosity
Data-driven closure model
System simulation with SAM-ML
3. Improve predictive capability
Conservation Equations
Data-driven closure model
2. Implementation
Coupling data-driven closure with SAM's multi-D module

Figure 1.1: Framework of integrating ML-based closure with nuclear system code for advanced reactor safety analysis.

$$\rho c_p \frac{\partial T}{\partial t} + \rho c_p \boldsymbol{V} \cdot \nabla \mathrm{T} - \nabla \cdot (k \nabla \mathrm{T}) - q''' = 0, \tag{3}$$

where $\boldsymbol{V}$ is velocity vector, p is pressure, $\boldsymbol{\tau}$ is stress tensor, c_p is specific heat, T is temperature, k is thermal conductivity, and q''' is heat source.

As a MOOSE (multiphysics object-oriented simulation environment)-based application [5], SAM uses a finite element method (FEM) to obtain the discretized equations. Instead of directly solving the PDEs, SAM solves the so-called "weak form" of the PDEs. Using FEM, the weak forms of the discretized equations are obtained by multiplying the governing equations by a test function, ψ, and then integrating over the domain to get the residual R of the equation:

$$R'_c = \left(\frac{\partial \rho}{\partial t}, \psi \right) + (-\rho \boldsymbol{V}, \nabla \psi) + \langle \rho \boldsymbol{V} \cdot n, \psi \rangle \tag{4}$$

$$R'_m = \left(\rho \frac{\partial \boldsymbol{V}}{\partial t}, \psi \right) + (\rho \left(\boldsymbol{V} \cdot \nabla \right) \boldsymbol{V}, \psi) + (\nabla \mathrm{p}, \psi) + (-\rho g, \psi) + (\mu \nabla \boldsymbol{V}, \nabla \psi) + \rho \boldsymbol{V} \cdot n, \psi \rangle + \langle -\mu \nabla \boldsymbol{V} \cdot \hat{n}, \psi \rangle \tag{5}$$

$$R'_e = \left(\rho c_p \frac{\partial T}{\partial t}, \psi \right) + (\rho c_p \boldsymbol{V} \cdot \nabla \mathrm{T}, \psi) + (k \nabla \mathrm{T}, \nabla \psi) + \langle -k \nabla T \cdot \hat{n}, \psi \rangle - \left(-q''', \psi \right), \tag{6}$$

where (,) denotes a volume integral and ⟨ , ⟩ a surface integral. The discretized "weak form" PDEs are then solved with the Jacobian-free Newton-Krylov (JFNK) method provided by the MOOSE framework.

Although SAM is not intended to be a CFD tool, turbulence effects must be taken into account to obtain physically plausible and numerically stable solutions. In SAM's multi-D module, when turbulence effects are considered, the molecular viscosity μ is replaced with the effective viscosity μ_{eff}. The effective viscosity includes the contributions from both the molecular viscosity and turbulent viscosity effects:

$$\mu_{eff} = \mu + \mu^t, \tag{7}$$

where μ^t is the turbulent viscosity.

Prior to this SAM-ML implementation, a zero-equation turbulence model based on Prandtl's mixing length theory was implemented in SAM :

$$\mu^t = \rho L_{mix}^2 \sqrt{2 \sum_{i,j=1}^{i,j=3} S_{ij} S_{ij}} \tag{8}$$

where $S_{ij} = \boldsymbol{S} = \frac{1}{2}\left[\nabla \boldsymbol{V} + (\nabla \boldsymbol{V})^T\right]$ is the rate-of-strain tensor, and L_{mix} is the mixing length that needs to be specified in the simulation input. Such a model has limited performance in thermal stratification problems in open volumes [3]. The more complicated RANS-based turbulence model, such as the $k - \epsilon$ model, is currently not considered in SAM due to its fine-mesh requirement and high computational cost. Instead, we aim to develop a coarse mesh data-driven turbulence closure that can learn from the fine mesh $k - \epsilon$ model.

1.3 CFD simulation of SFR loss-of-flow transients

In this report, we demonstrate the applicability of the proposed framework in a case study of the protected loss-of-flow (PLOF) transients, a key scenario in safety evaluation of a SFR design. The PLOF transient assumes the complete loss of forced flow due to multiple component failures. SFRs are typically designed with high inherent safety, and can maintain sufficient core cooling even in this extreme scenario, where the reactor-protection system is assumed to have functioned and shut down the core immediately, with only the decay heat remaining to be removed. All pumps and all emergency-powered safety systems are assumed to fail. This leaves, as the only heat removal path, the emergency heat-removal system, for which flow is maintained by natural convection alone. Any other intermediate loops, balance of plant, etc., are not assumed to provide any heat-removal capability.

To make the scenario investigated here as relevant to real-world applications as possible, the transient was planned using the preconceptual design report of the Advanced Burner Test Reactor (ABTR) [6], a pool-type SFR design. A baseline PLOF scenario was investigated numerically for the ABTR using SAM's 1-D module. The 1-D simulation solution was used to provide inlet conditions for the CFD simulation as the baseline training transient, which is denoted as Transient 1 in the remainder of this paper. By perturbing the inlet conditions of Transient 1, two other transients, denoted as Transients 2 and 3, were simulated by CFD to test the generality of the trained ML model. The inlet conditions of mass flow rate Q and temperature T of the three transients are depicted in Figure 1.2. Compared to Transient 1, Transient 2 has a slower mass flow rate change and a smaller temperature change, while Transient 3 has a faster mass flow rate change and a larger temperature change.

The computation domain of the CFD simulation is a simplified cylindrical tank that represents the ABTR hot pool, as the flow mixing and thermal stratification in this region is of high importance to the safety of the SFR. The tank's diameter is 1.95 m and its height is 7.74 m. The tank inlet is centered at the bottom origin, with a diameter of 0.43 m; its outlet is located at a height of 4.1 m, and is a narrow ellipse representing the inlet to a heat exchanger. The outlet window is 20 cm high and 2.175 m wide. It was assumed that there were two heat exchanger loops on opposite sides of the tank, resulting in a symmetrical problem. This characteristic was exploited in the simulation by using a symmetry plane along the center of the domain, opposite the outlet. The outlet was also extruded to help prevent backflow. The inlet is flush with the bottom of the tank. At steady state, the inlet mass flow rate is 632 kg/s and the inlet temperature is taken to be 783.15 K, representing half of the primary system mass flow rate and the core outlet temperature.

The computation domain of the CFD simulation is a simplified cylindrical tank that represents

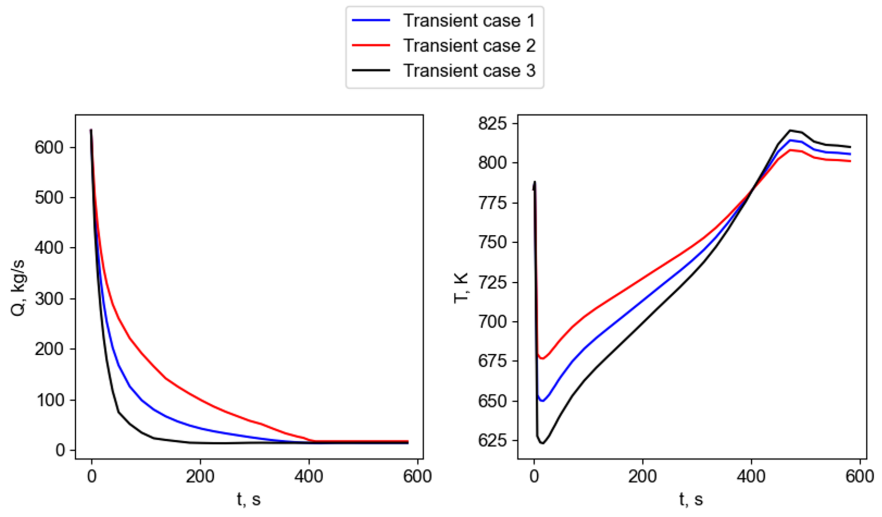


Figure 1.2: Three loss-of-flow transients for DCNN-LSTM training and performance evaluation.

the ABTR hot pool, as the flow mixing and thermal stratification in this region is of high importance to the safety of the SFR. The tank's diameter is 1.95 m and its height is 7.74 m. The tank inlet is centered at the bottom origin, with a diameter of 0.43 m; its outlet is located at a height of 4.1 m, and is a narrow ellipse representing the inlet to a heat exchanger. The outlet window is 20 cm high and 2.175 m wide. It was assumed that there were two heat exchanger loops on opposite sides of the tank, resulting in a symmetrical problem. This characteristic was exploited in the simulation by using a symmetry plane along the center of the domain, opposite the outlet. The outlet was also extruded to help prevent backflow. The inlet is flush with the bottom of the tank. At steady state, the inlet mass flow rate is 632 kg/s and the inlet temperature is taken to be 783.15 K, representing half of the primary system mass flow rate and the core outlet temperature.

The computation domain is depicted in Figure 1.3. When evaluating the performance of the ML model, we extract CFD results to make qualitative and quantitative comparisons with the model's prediction. In this work, the results at two planes were extracted for qualitative comparison, while the results at two lines (one vertical line along the vertical center of the domain, one horizontal line pointing to the outlet of the domain) were extracted for quantitative comparison.

The commercial finite-volume CFD code STAR-CCM+ [7] was used to solve the transient problem. Sodium was the working fluid, with temperature-dependent properties except a constant specific heat value of $c_p = 1286\, J/kg \cdot K$. The sodium density ρ, dynamic viscosity μ and thermal conductivity κ are evaluated as

$$\rho = 1015.03 - 0.23393T - 3.05 \times 10^{-6}T^2, \tag{9a}$$

$$\mu = \exp\left(-6.4406 - 0.3958 \ln T + \frac{556.835}{T}\right), \tag{9b}$$

$$\kappa = 124.67 - 0.11381T + 5.5226 \times 10^{-5}T^2 - 1.1842 \times 10^{-8}T^3. \tag{9c}$$

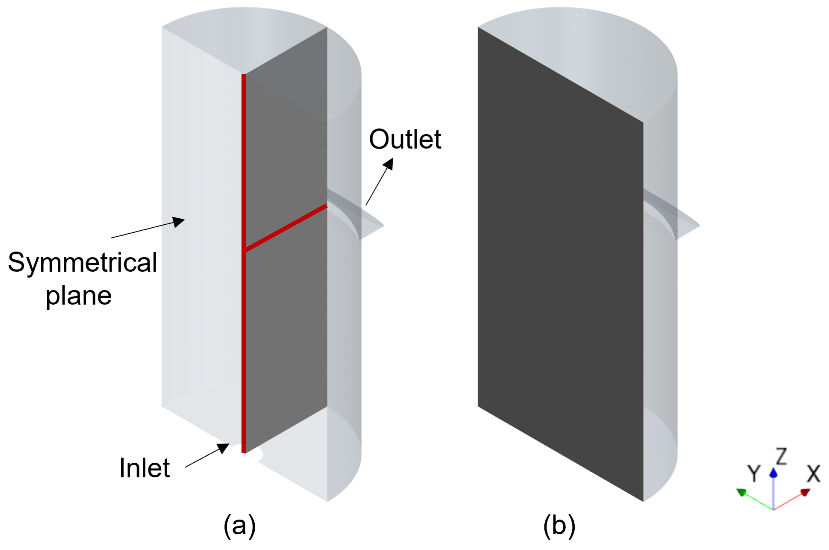


Figure 1.3: Computation domain of half symmetrical cylinder with data extracted for ML model performance evaluation at (a) outlet plane and two lines; (b) symmetrical plane.

Gravity acts in the –Z direction, i.e., in the opposite direction from the inlet flow. The inlet conditions of mass flow rate and temperature during the transient as described in Figure 1.2 were implemented as tables with linear interpolation between points.

The top of the tank was treated as a slip wall. In an actual reactor, there is a cover gas, typically argon, to allow for space to account for changes in the sodium volume and pressure. This gas was not included, in part to allow for simpler physics for the ML model. However, it is noted that multiphase simulations of the tank were also performed using a Volume-of-Fluid model. No substantial changes in the overall flow behavior were encountered, primarily just a small raising of the interface in the center of the domain in the vicinity of the core outlet jet, and impacts on the outlet temperature were small. Thus the slip-wall simplification was deemed acceptable for the current work.

A mesh of roughly 1.05 million trimmed hexahedral cells was employed, with refinement in roughly the inner half of the tank to better resolve the jet emerging from the core. Two prismatic layers were used at the wall. A 2nd-order segregated solver was used for the flow and energy equations. The RANS-based Realizable k-ε Two-Layer model was used to model turbulence in the system. This model was used along with the thermal-stratification model for the buoyancy production of dissipation. The adopted k-ε turbulence model represents a compromise between accuracy and robustness, and has been shown to be reasonably accurate in prior simulations that featured buoyant jets and erosion of stratified layers [8]. A Reynolds-Stress model was also tested, but was shown to be similar enough to the base turbulence model that it did not warrant the additional cost. A turbulent Prandtl number of 2 was employed, which is common in simulations of liquid-metal flows. The turbulent viscosity μ^t that was computed from the obtained turbulent kinetic energy k and turbulent dissipation rate ε was then used as the training target of the ML model:

$$\mu^t = \rho C_\mu \frac{k^2}{\varepsilon}, \tag{10}$$

where ρ is the density of the liquid sodium, which is a function of temperature; and C_μ is a coefficient with the fixed value of 0.09.

Since a PLOF transient is assumed to start from nominal conditions, a steady-state run was performed to obtain the initial flow and temperature fields for all three transients. After that period, a variable time stepper was used to keep the mean Courant–Friedrichs–Lewy condition below 0.5, with a maximum time step of 0.1 s and a maximum change factor of 2 per time step. The transients were run for 600 s. To ensure that the time-sequence data have equal weight in training the model, we extracted the simulation results with a uniform time step of 0.2 s, so for a 600 seconds transient, there are 3000 results. As the ML model considers both temporal and spatial information from the data, we further combine the 3000 snapshots into 300 samples, with each sample consisting of a consecutive sequence of 10 time steps. The ML is then trained with these samples.

2 Deep neural network for coarse-mesh turbulence prediction

The goal of this work is to develop a coarse-mesh closure for the isotropic turbulent viscosity to support the safety modeling of advanced nuclear reactors, which often involve reactor transients with minutes or hours long with dynamic changes in inlet and boundary conditions. The desired DNN model for this work should be a field-to-field mapping that can be efficiently implemented into nuclear system code. In this work, we chose be a field-to-field mapping that takes the local flow feature fields $\boldsymbol{Q}_t = [\boldsymbol{q}_t^1, \boldsymbol{q}_t^2, \cdots, \boldsymbol{q}_t^n]$ as inputs and is informed by previous time-step information $\mathcal{L}_{t-1}$ to predict the turbulent viscosity field $\boldsymbol{\mu}_t^t$ at time step t:

$$f\left(\boldsymbol{Q}_t,\ \mathcal{L}_{t-1}\right) = \ \boldsymbol{\mu}_t^t, \tag{11}$$

where $\boldsymbol{q} \in \mathbb{R}^{X \times Y \times Z}$ is the flow-feature field of the computation domain with the dimensionality of $X \times Y \times Z$(depending on the mesh size), $\boldsymbol{\mu}_t^t$ is the turbulent viscosity field with the same dimensionality, and $\mathcal{L}_{t-1}$ is the latent space information of previous time steps with a compressed dimensionality.

The proposed field-to-field mapping requires more sophisticated neural network architecture instead of the classical feedforward neural network, even though the latter had successful applications in closure development [9]. In this report, we investigated three different neural network architectures and evaluated their performance in the case study.

2.1 POD-LSTM: long-short-term-memory network based on proper orthogonal decomposition

As the problem we aim to solve is reactor transients that involve temporal information, the first neural network architecture we investigated is the recurrent neural network (RNN). In RNN, the result of time step t depends not only on input x_t at time step t, but also on the hidden state h_{t-1} at the previous time step. Specifically, we utilized the long-short-term-memory (LSTM) netwok [10], a modern recurrent neural network architecture to construct the field-to-field mapping.

The LSTM network contains a memory cell and multiple gates to control the information flow of the cell, as depicted in Figure 2.1.

In Figure 2.1, f stands for the forget gate, which determines whether to erase the memory cell; i stands for the input gate, which determines whether to write the results into the memory cell; o stands for the output gate, which determines whether to export the current results stored in the memory cell; and g stands for the gate that couples with the input gate to determine the specific value stored in the memory gate. The information flow within the LSTM unit is computed together with an extended trainable weight matrix W, and through a different non-linear activation function:

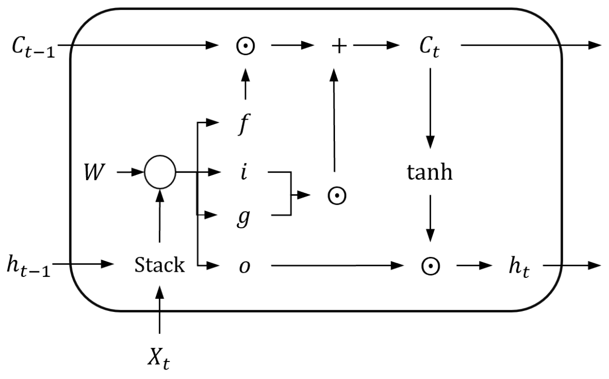


Figure 2.1: Information flow in a LSTM unit.

$$\begin{aligned} f_t &= \sigma\left(W_{xf} * X_t + W_{hf} * \mathcal{L}_{t-1} + W_{cf} \odot C_{t-1}\right) \\ i_t &= \sigma\left(W_{xi} * X_t + W_{hi} * \mathcal{L}_{t-1} + W_{ci} \odot C_{t-1}\right) \\ g_t &= tanh(W_{xg} * X_t + W_{hg} * \mathcal{L}_{t-1}) \\ o_t &= \sigma\left(W_{xo} * X_t + W_{ho} * \mathcal{L}_{t-1} + W_{co} \odot C_t\right) \\ C_t &= f_t \odot C_{t-1} + i_t \odot g_t \\ \mathcal{L}_t &= o_t \odot tanh(C_t), \end{aligned} \quad , \tag{12}$$

where σ stands for sigmoid activation function and $\odot$ stands for matrix elementwise multiplication.

It is not practical to directly input the high dimensional training data into the LSTM network. In this work, we use the proper orthogonal decomposition (POD) to identify a subspace of the high dimensional training data, which serves as input to the LSTM network. In this work, we employ the singular value decomposition (SVD), an algorithm with superior numerical stability, to identify the principal components of the given physical field M during the full transient:

$$\boldsymbol{M} = \boldsymbol{U}\boldsymbol{\Sigma}\boldsymbol{V}^T \ , \tag{13}$$

where $\boldsymbol{\Sigma}$ is a diagonal matrix whose entry values corresponding to the root of the eigenvalues of $\boldsymbol{M}\boldsymbol{M}^T$; bmU is a unitary matrix, whose column vectors are termed the PCs, with the equal size of the column vector of $\boldsymbol{M}$, and $\boldsymbol{V}$ is a square unitary matrix. We retain the first k PCs from $\boldsymbol{U}$ to form a matrix $\boldsymbol{\Phi}$. $\boldsymbol{\Phi}$ is chosen in a way that it can account for at least 90% of the total variance of $\boldsymbol{M}$. $\boldsymbol{\Phi}$ is then used to map the column vector $\boldsymbol{m}$ to the PC subspace $\boldsymbol{y}$, so its dimensionality is significantly reduced (in general, from a few thousand mesh grids to 10-30 PCs). Moreover, the results on the PC subspace can be recovered to its original physical space through $\boldsymbol{\Phi}$. Such a dimensionality reduction process is illustrated in Figure 2.2.

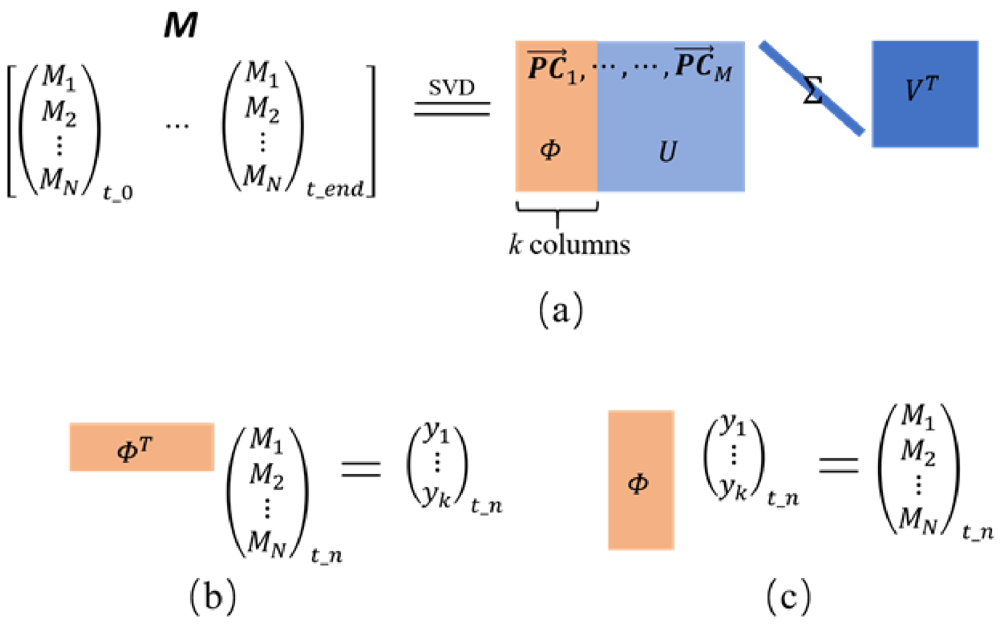

M
SVD
$\overrightarrow{PC}_1, \cdots, \cdots, \overrightarrow{PC}_M$
Φ
U
Σ
V^T
k columns
(a)
Φ^T
(b)
Φ
(c)

Figure 2.2: (a). Obtaining principal components through SVD; (b) Mapping physical field from physical space to principal component subspace; (c) Recover results on principal component subspace to original physical space.

2.2 DCNN: densely connected convolutional neural network

Convolutional neural networks (CNNs) [11] have demonstrated superior performance in the data-driven modeling of spatial-temporal dynamic systems [12]. For this reason, the backbone of the field-to-field mapping is chosen to be CNN, which is also informed by the recurrent neural network (RNN) to learn the temporal information.

There are additional advantages for developing the mapping based on CNN. First, convolution can serve as a numerical approximator of differential operators, as Dong et al. [13] proved that convolution operation has a profound mathematical relationship with differentiation operators. In this sense, CNN can be used to implicitly learn derivatives of an input physical quantity, so these derivative terms are not needed to be explicitly included as inputs. Such an adoption significantly simplifies the problem.

Furthermore, the RANS equations that describe the thermal-fluid system obey Galilean invariance, which includes rotation invariance, translation invariance, and uniform motion invariance. In this work, the goal is to model the isotropic turbulent viscosity, so the rotational invariance is insignificant under the isotropic assumption. The derivative terms of physical quantities satisfy the invariance of translation and uniform motion [14]. So in the context of this work, the CNN architecture is consistent with the Galilean invariance.

CNN is based on multiple convolutional kernels that slide along the input fields to produce lower-dimensional feature maps. The convolution operation on fields $\boldsymbol{Q}$ with kernel $\boldsymbol{K}$ can be expressed as follows:

$$(\boldsymbol{Q} * \boldsymbol{K})(x, y, z) = \sum_m \sum_n \sum_l \boldsymbol{Q}(x-m,\ y-n,\ z-l)\, \boldsymbol{K}(m, n, l). \tag{14}$$

The convolution product depends on the size (denoted by k) and the weights of the kernel, as well as the convolution operation's stride s and padding p.

Compared to feedforward neural networks, the weight-sharing and sparse connection features of CNN are advantageous for learning from 3-D data. Furthermore, previous studies reveal that the convolution operation has a profound mathematical relationship with differentiation operators [13]. Supported by this relationship, we do not need to extract gradients of flow fields as inputs when training the neural network; this feature significantly simplifies the problem compared to the classical feedforward neural-network approach.

In this work, we adopt the densely connected convolutional neural network (DCNN) [15], an advanced CNN architecture, to learn the spatial information from the 3-D CFD data. Compared to traditional CNN, DCNN consists of multiple dense blocks of convolutional layers; within each block, there are direct connections from any layer to all subsequent layers. Such a configuration allows better information and gradient flow between the layers of the network, so that training is easier compared to a traditional CNN. The difference between CNN and DCNN is depicted in Figure 2.3.

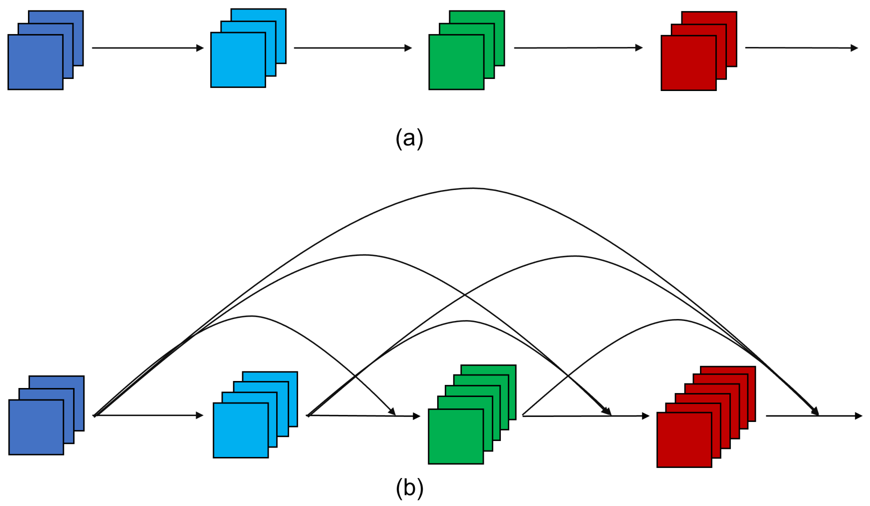


Figure 2.3: Difference between (a) convolutional layers and (b) densely connected convolutional layers, using a 2-D feature map as an example.

The designed DCNN network for spatial-information learning takes five flow-feature fields as inputs, i.e., three velocity components V_x, V_y, V_z, pressure p, and temperature T. The output of the network is the turbulent viscosity field $\boldsymbol{\mu}^t$. Inspired by a successful application in porous media flow [16], we adopted a similar DCNN structure that consists of an odd number of dense blocks along with an encoder-decoder layer setup, as depicted in Figure 2.4.

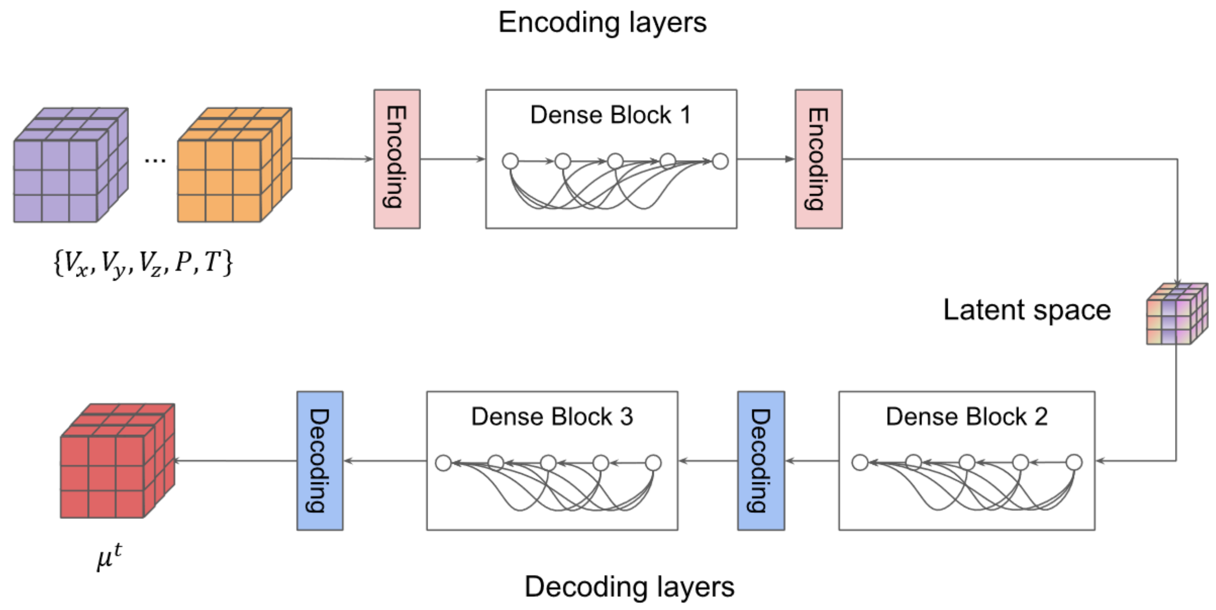


Figure 2.4: DCNN structure for coarse-mesh turbulent viscosity prediction.

The encoding/decoding layer consists of two convolutional layers connected by a rectified linear unit (ReLU) layer. In the encoding layer, the first convolutional layer has $(k, s, p) = (1,1,0)$, whose purpose is to limit the number of feature maps for a more concise network. The second convolutional layer has $(k, s, p) = (3,2,1)$, which will reduce the dimensionality of the feature map by a factor of 2. The decoding layer has a similar setup, with the difference that the second convolutional layer is a transpose-convolution operation that will increase the dimensionality of the feature map by a factor of 2, sot that the ultimate output has the same dimensionality as the input flow fields. We also applied a dropout layer after each convolutional layer to serve as a regularization term to mitigate the overfitting issue [17]. The dropout layer will randomly zero some of the elements of its input tensor with probability p, using samples from a Bernoulli distribution during the training process. Meanwhile, the outputs are also scaled by a factor of $1/(1-p)$.

2.3 DCNN-LSTM: combining DCNN with LSTM

In this work, we designed a novel neural network architecture DCNN-LSTM based on the concept of convolutional recurrent neural networks [18]. Our design aims to efficiently learn the spatial-temporal CFD data with a combination of DCNN and LSTM. Based on the DCNN structure, the LSTM units take the latent space $\mathcal{L}$ obtained from DCNN as inputs; the information flow follows the one depicted in Figure 2.1, but with a series of convolution operations instead of matrix multiplication:

$$\begin{array}{c} f_t = \sigma\left(W_{xf} * X_t + W_{hf} * \mathcal{L}_{t-1} + W_{cf} \odot C_{t-1}\right) \\ i_t = \sigma\left(W_{xi} * X_t + W_{hi} * \mathcal{L}_{t-1} + W_{ci} \odot C_{t-1}\right) \\ g_t = tanh(W_{xg} * X_t + W_{hg} * \mathcal{L}_{t-1}) \\ o_t = \sigma\left(W_{xo} * X_t + W_{ho} * \mathcal{L}_{t-1} + W_{co} \odot C_t\right) \\ C_t = f_t \odot C_{t-1} + i_t \odot g_t \\ \mathcal{L}_t = o_t \odot tanh(C_t), \end{array} , \tag{15}$$

where $*$ stands for convolution operations.

By taking $\mathcal{L}$ as the temporal input, DCNN-LSTM is able to learn the temporal information of the CFD transient data with a concise LSTM module, as the $\mathcal{L}$ has a compressed size of the original 3-D CFD data. The full DCNN-LSTM architecture is depicted in Figure 2.5. Such a design ensures efficient learning for both spatial and temporal information from the 3-D transient CFD data.

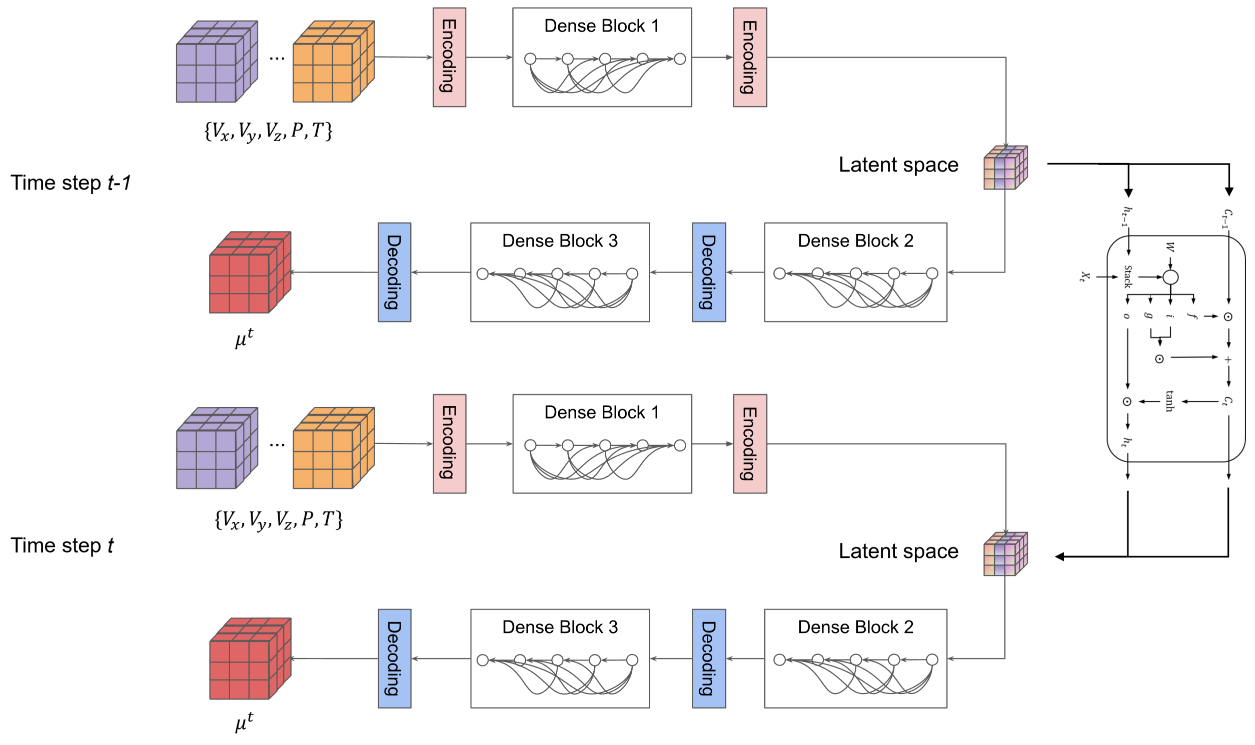


Figure 2.5: DCNN-LSTM architecture for coarse-mesh turbulent viscosity prediction during reactor transients.

3 Training machine learning-based closure

In this section, a complete workflow of training ML-based closure for coarse mesh turbulence modeling is discussed.

3.1 Data preprocessing

The original CFD case used more than 1.05 million nonuniform meshes for simulation. The physical quantity fields defined on such a high-resolution nonuniform mesh cannot be directly used for training the proposed DCNN-LSTM model, for two reasons. First, nonuniform mesh means the data from each mesh cell do not have equal weight, making it difficult to train a neural network model that treats each element with equal weight. Second, the high-resolution setup is not compatible with the desire for a constitutive relation for a coarse-mesh 3-D module in system code. To overcome these two issues, the k-Nearest-Neighbor (kNN) algorithm provided by the open-source ML library scikit-learn [19] was utilized to convert the original CFD results to uniform coarse-mesh data. In the conversion, we considered only the cylindrical tank, and the outlet extrusion was not included in the converted data.

Given a point A in the new uniform coarse-mesh setup, the kNN algorithm is able to find a predefined number of points in the nonuniform fine mesh closest in distance to A, and then predict the value of A through a distance-weighted average with the values of these neighboring points. We confirmed the validity of this data preprocessing procedure by qualitatively and quantitatively comparing the original CFD results and the processed kNN results. An example of three key physical quantities, i.e., the vertical velocity component V_z, temperature T, and turbulent viscosity μ^t at $t = 200.0$ s, is depicted in Figures 3.1 and 3.2. The kNN results are found to be in very good agreement with the original CFD results, with the largest discrepancy (< 10 K) observed in the temperature in the horizontal line pointing to the outlet center. Such an observation confirms that the data accuracy can be preserved when converting the fine-mesh nonuniform setup to a coarse-mesh uniform one.

As a field-to-field mapping process, the DNN models investigated in this work take 3-D PyTorch tensors as input and generates a 3-D tensor as output with the same dimension. In this work, the processed kNN results were padded with zero outside the semi-cylinder computation domain so that each physical scalar field can be converted to a regular 3-D tensor. Combining the kNN and padding, we were able to convert the fine-mesh nonuniform CFD results to coarse-mesh uniform tensors with dimensions of $32\times64\times128$ for further model training.

3.2 Baseline model training

With the coarse-mesh training data ready, the next step is to obtain a baseline model from the three neural network architectures, i.e. POD-LSTM, DCNN, and DCNN-LSTM. In this report, we chose to train the model with data from Transient 1 only and select one neural network architecture with highest accuracy for further study. The results of the two other transients as test cases to

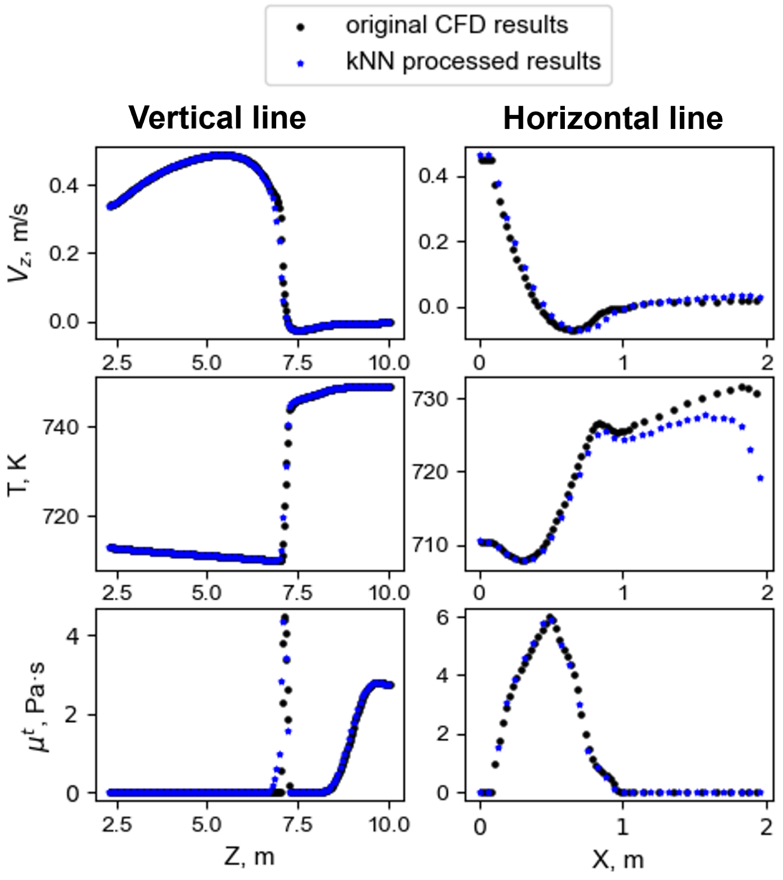


Figure 3.1: Quantitative comparison between original CFD results and processed kNN results, $t = 200$ s.

evaluate the model's generalization capability for "extrapolating" inlet conditions. For the results of Transient 1, we further decomposed the full dataset into two parts: a training dataset with 70% of the full dataset, and a testing dataset with the remaining 30%. The DCNN-LSTM model was trained with the training dataset, using an error function $\ell(f_\theta(x), y)$ based on the root mean square error (RMSE) along with an additional regularization term to mitigate the overfitting of the model:

$$\ell(f_\theta(x), y) = \sqrt{\frac{1}{m}\sum_{i=1}^{m}(\hat{y}_i - y_i)^2} + \lambda \boldsymbol{w}^T \boldsymbol{w}, \tag{16}$$

where $\boldsymbol{w}$ is the learnable weights of the model, and λ is the weight decay factor.

It is well-known that the hyperparameters of a neural network can have a significant influence on its performance. In this work, we performed a comprehensive hyperparameter tuning process for the DNN models with optimal performance. The first step is to select a baseline model with

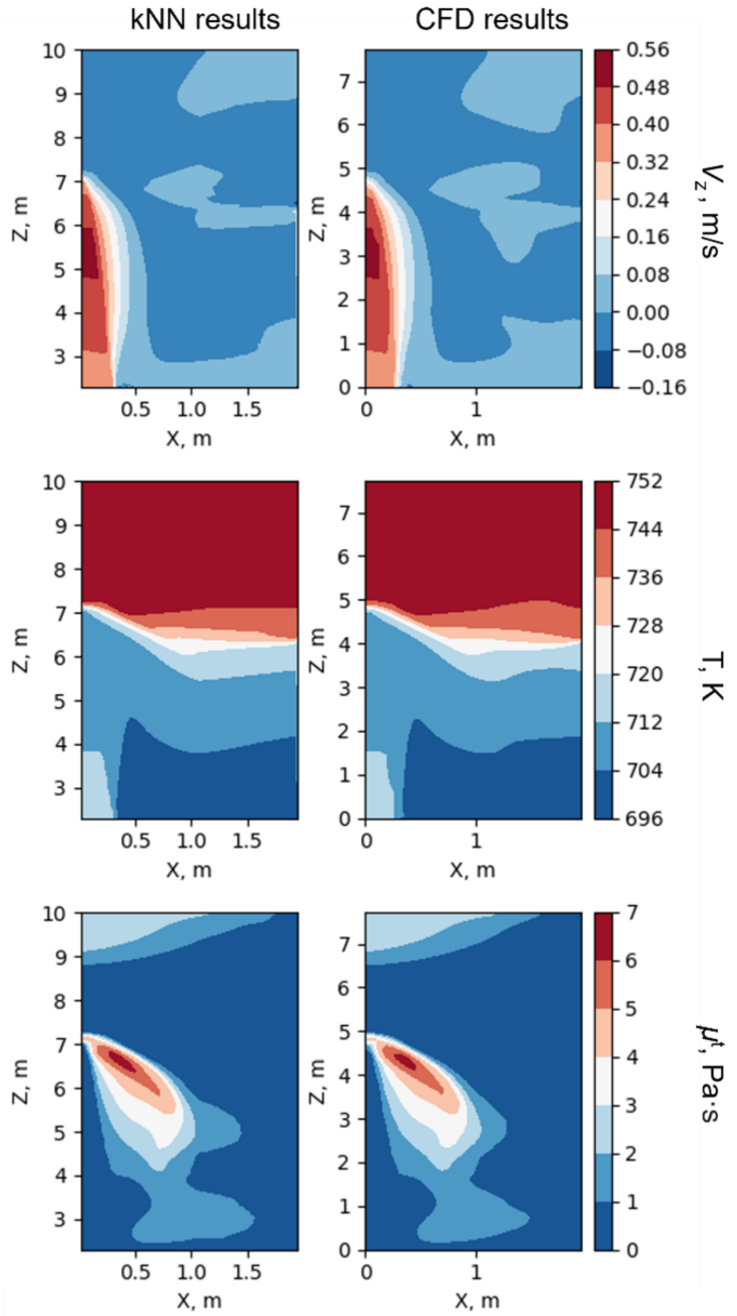


Figure 3.2: Qualitative comparison between original CFD results and processed kNN results at outlet plane, $t = 200$ s.

reasonably low RMSE on the testing dataset by manually perturbing a few key hyperparameters, including learning rate, regularization parameter, and batch size. The RSMEs of a few case studies are depicted in Figures 3.4, 3.3, and 3.5.

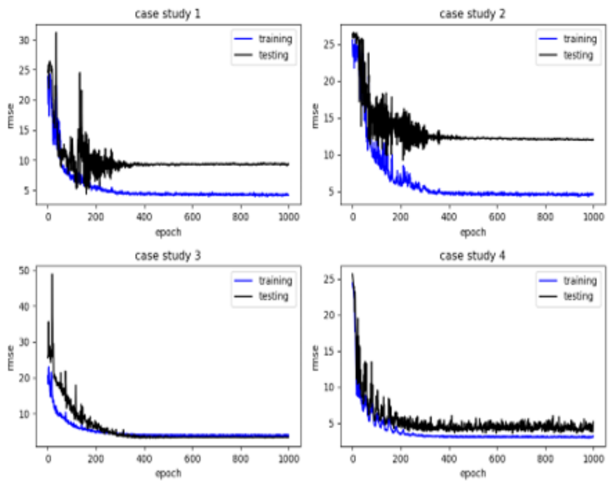


Figure 3.3: Training baseline DCNN model with different hyperparameter setups.

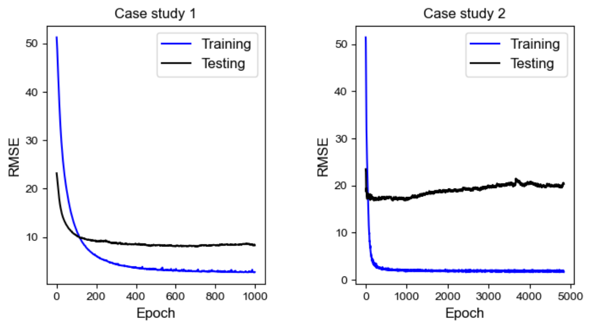


Figure 3.4: Training baseline POD-LSTM model with different hyperparameter setups.

It can be found from the figures that for all three DNN models, the RMSE decreases with the training process, but the RMSEs of DCNN-LSTM show a fluctuating pattern. Such a fluctuating pattern is typical in training the LSTM model, as training a model involving LSTM module is usually more challenging than the normal feedforward network. The performance of the three DNN models in testing dataset is depicted in Figure 3.6. It can be found that all three models can capture the very dynamic pattern change of turbulent viscosity during the transient, but overall, the DCNN-LSTM model has better agreement with the original CFD data. Such an observation is consistent with the fact that the architecture of DCNN-LSTM can learn from both the spatial information and temporal information from the CFD data. For this reason, the DCNN-LSTM model is chosen as the baseline model for further investigation.

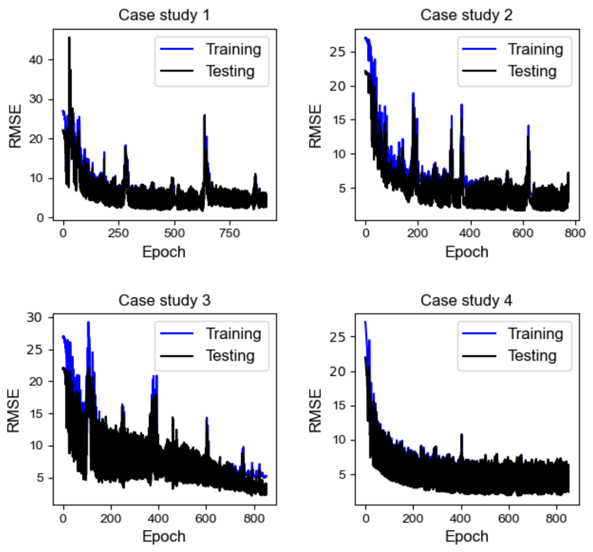


Figure 3.5: Training baseline DCNN-LSTM model with different hyperparameter setups.

3.3 Hyperparameter optimization of DCNN-LSTM

With the selected baseline model of DCNN-LSTM, a more comprehensive hyperparameter optimization was performed on a larger number of hyperparameters using DeepHyper [20], an open-source scalable automated ML package.

Following the Bayesian optimization framework, DeepHyper uses an asynchronous model-based search to obtain an optimized set of hyperparameters with minimal RMSE on the testing dataset. In DeepHyper, a surrogate model S that takes the hyperparameter set as input and RMSE as output is constructed. This surrogate model is an approximation of the true RMSE on the whole hyperparameter space. For a hyperparameter set s, S predicts the mean $\mu(s)$ and standard deviation $\sigma(s)$ of the RMSE. In this work, we chose DeepHyper's built-in random forest algorithm to construct the surrogate model, as it demonstrated superior performance compared to random search and genetic algorithm-based search [21]. Furthermore, the random-forest algorithm can handle discrete parameters (such as batch size) conveniently.

An acquisition function can be defined on the basis of $\mu(s)$ and $\sigma(s)$ for the Bayesian optimization. In this work, we chose the lower-confidence-bound acquisition function:

$$A(s) = \mu(s) - \beta\sigma(s), \tag{17}$$

where $\beta > 0$ is a tradeoff parameter. Starting from a prior surrogate model S_{prior} based on a number of initial hyperparameter samples, DeepHyper iteratively identifies new samples to update S, so the approximation to the true RMSE on the hyperparameter space can be improved. The intuition behind the adopted acquisition function involves weighing the relative importance of the

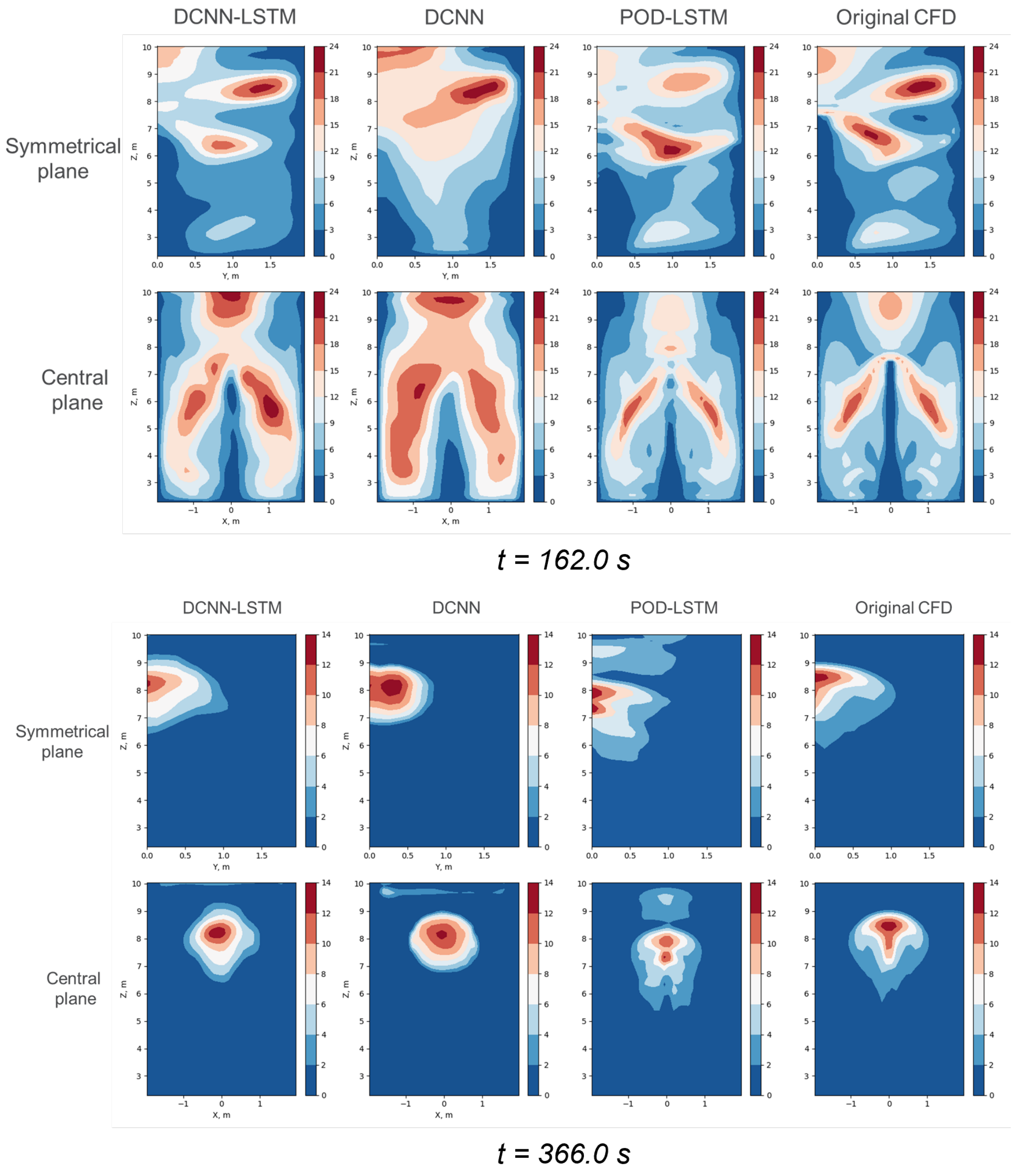


Figure 3.6: Comparison of three DNN models on testing dataset.

surrogate's mean and uncertainty to achieve a balance between exploitation and exploration of the hyperparameter space that is controlled by β. In this work, we chose $\beta = 1.96$ to encourage exploration. The whole Bayesian optimization process is illustrated in Figure 3.7.

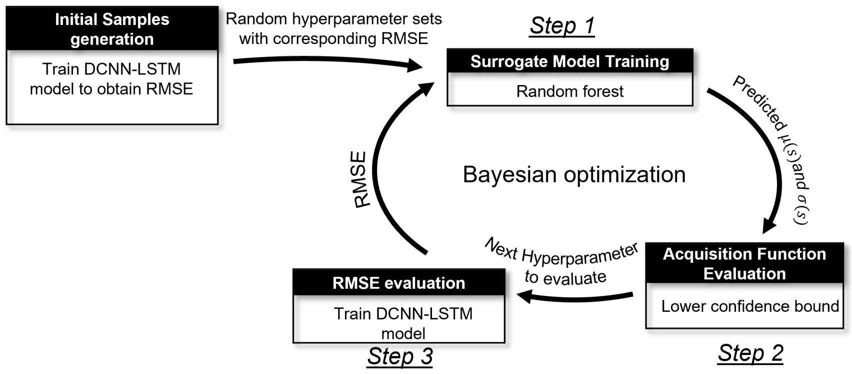

Initial Samples generation
Train DCNN-LSTM model to obtain RMSE
Random hyperparameter sets with corresponding RMSE
Step 1
Surrogate Model Training
Random forest
Predicted $\mu(s)$ and $\sigma(s)$
Bayesian optimization
RMSE
Acquisition Function Evaluation
Lower confidence bound
Step 2
Next Hyperparameter to evaluate
RMSE evaluation
Train DCNN-LSTM model
Step 3

Figure 3.7: Bayesian optimization using DeepHyper.

Considering the relatively large computational cost for training hundreds of neural networks, we performed the Bayesian optimization with 20% of the training data and ran for only 100 epochs. The optimization was performed with nine hyperparameters, i.e., batch size, dropout rate, growth rate of dense block, initial features of dense block, learning rate, weight-decay factor, and the number of dense layers in each of the three dense blocks. A total of 246 evaluations were performed on the Argonne Leadership Computing Facility's Theta cluster. Through the Bayesian optimization, we found that three hyperparameters have the most significant impact on the RMSE of the testing dataset, i.e., the dropout rate, learning rate, and weight decay. The pairwise plots and the marginal distributions of these three hyperparameters are depicted in Figure 3.8.

With the optimized hyperparameters, we then trained the DCNN-LSTM model on the full training dataset with 600 epochs. Besides RMSE, the R-square value is also calculated to evaluate the performance of the optimized model; this value describes the ratio between the variance explained by the DCNN-LSTM model and the total variance of the dataset, and can be computed as follows:

$$R^2 = 1 - \frac{\sum_{i=1}^{m} (\hat{y}_i - y_i)^2}{\sum_{i=1}^{m} (\bar{y}_i - y_i)^2}, \tag{18}$$

where $\bar{y}_i$ is the mean value of the output. An R^2 value close to 1 indicates very good regression of the model. As demonstrated in Figure 3.9, the optimized DCNN-LSTM model has lower RMSE compared to the baseline model.

3.4 Evaluation of the generalization capability of DCNN-LSTM

Compared to the effectiveness of training the DCNN-LSTM model, it is more important to evaluate the generalization capability of the trained network with cases of "extrapolating" inlet conditions, as the ultimate purpose of training the neural network model is to apply it to a variety of conditions

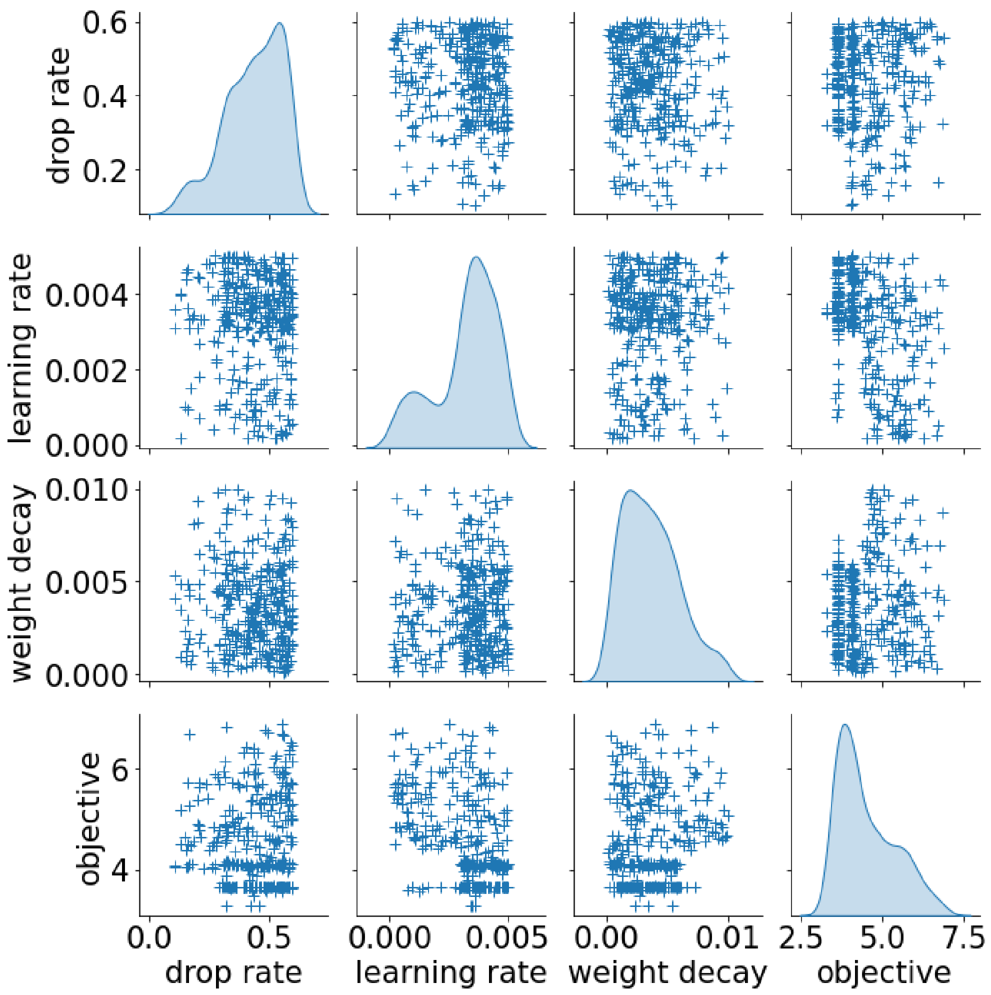


Figure 3.8: Pairwise plots and marginal distributions of DCNN-LSTM hyperparameter search using DeepHyper.

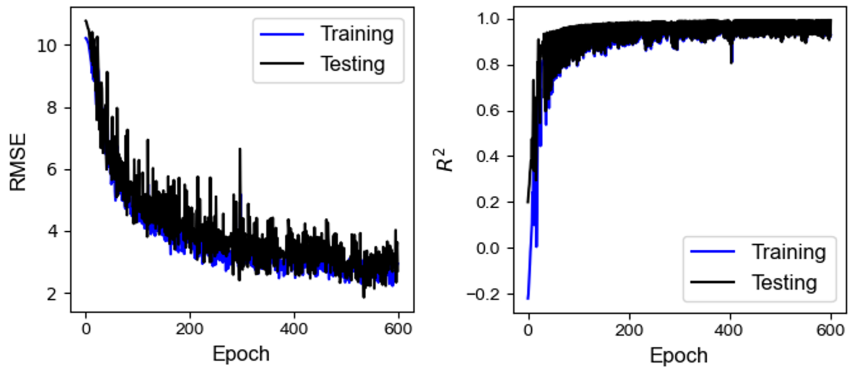


Figure 3.9: RMSE and R-square of optimized DCNN-LSTM training.

that do not have available CFD simulations. In this work, the generalization capability of the DCNN-LSTM model is tested with two other loss-of-flow transients with the inlet conditions perturbed from Transient 1, as shown in Figure 1.2.

In a similar manner to the training effectiveness evaluation, we obtain the sequence of the turbulent viscosity field for Transients 2 and 3. The snapshots at $t = 70.2$ s of both cases are used for comparison. The qualitative and quantitative comparisons are depicted in Figures 3.10-3.13.

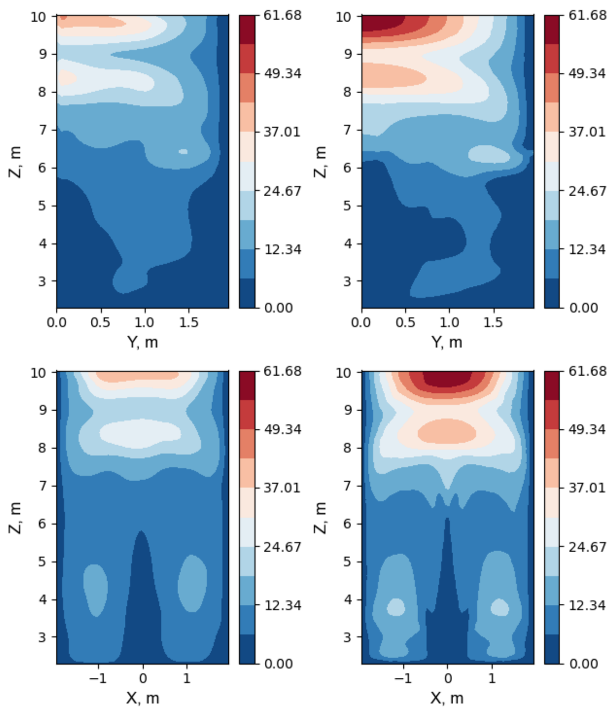


Figure 3.10: Qualitative comparison between DCNN-LSTM prediction and original CFD results at $t = 70.2$ s, Transient 2.

We can observe from the figures that compared to the results of Transient 1 at the same time step, Transients 2 and 3 showed distinctive turbulent viscosity fields, suggesting that a moderately perturbed inlet condition can cause significant differences in local flow fields. It can also be found that the turbulent viscosity field predicted by the DCNN-LSTM model is generally consistent with the CFD results in terms of trend and distribution. However, the prediction underestimated the turbulent viscosity for both cases. The generally consistent turbulent viscosity field indicates the model has generalization capability for "extrapolating" inlet conditions. But the underestimation suggests that the generalization capability is not good enough for the model to give predictions on "extrapolating" cases at the same accuracy as the training case.

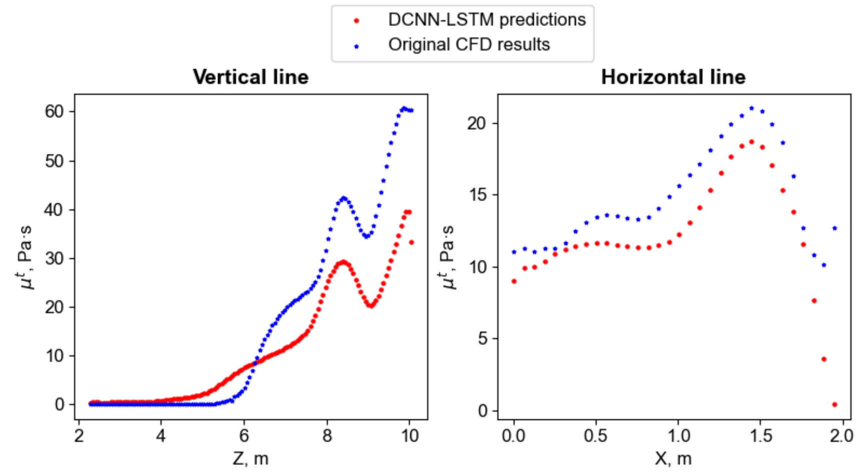


Figure 3.11: Quantitative comparison between DCNN-LSTM prediction and original CFD results at $t = 70.2$ s, Transient 2.

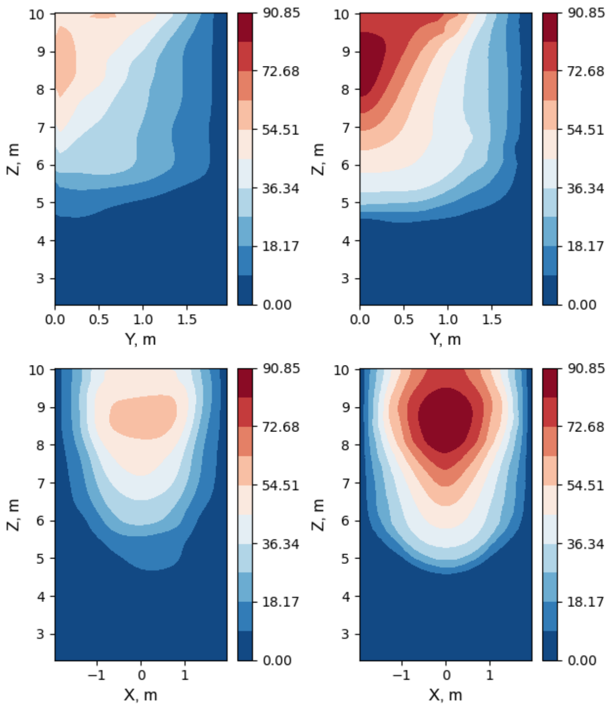


Figure 3.12: Qualitative comparison between DCNN-LSTM prediction and original CFD results at $t = 70.2$ s, Transient 3.

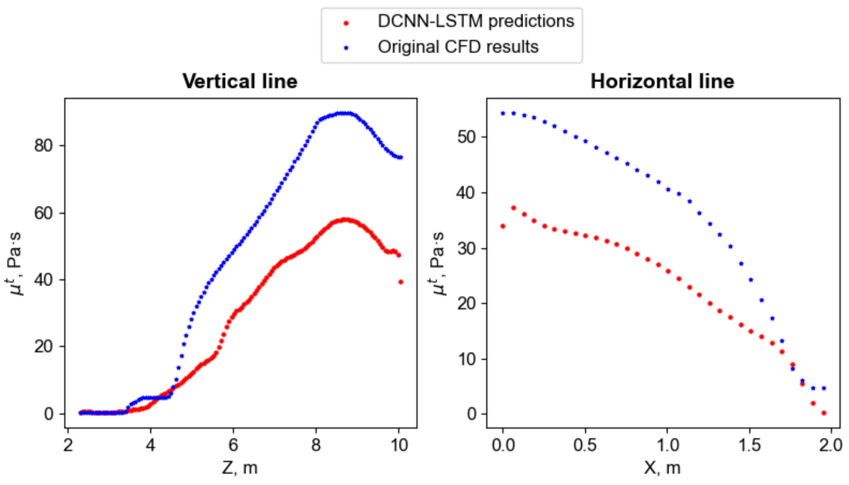


Figure 3.13: Quantitative comparison between DCNN-LSTM prediction and original CFD results at $t = 70.2$ s, Transient 3.

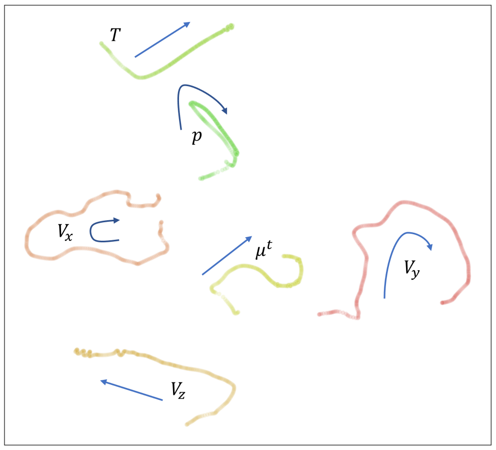
3.5 Explanation of DCNN-LSTM's generalization capability

We employed t-distributed stochastic neighbor embedding (t-SNE) [22] to study the intrinsic data similarity among the CFD results of the three transients. The t-SNE method is a popular method for exploring high-dimensional data; it deals with data in high-dimensional space and find a faithful representation of those data in a lower-dimensional space, typically 2-D or 3-D for visualization purposes. Through t-SNE, the data with high similarity are mapped into nearby points and data with high dissimilarity are mapped into distant points with high possibility. In this work, we use the scikit-learn library to perform the t-SNE.

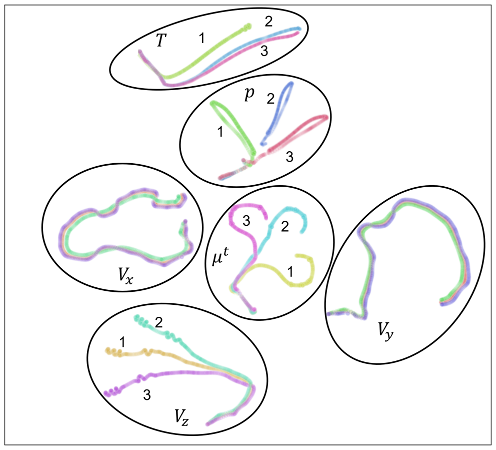
We used t-SNE to map the scalar fields of six physical quantities (five inputs and one output) into a 2-D map. To eliminate the scaling influence on the t-SNE map, all the physical quantities are normalized by the mean and standard deviation of the transient they belong to. Each physical field at a time snapshot is a 3-D tensor with dimensions $32 \times 64 \times 128$. These high-dimensional data are converted to a point in the 2-D t-SNE map for visualization purpose. The resulting maps are depicted in Figure 3.14.

It can be found from Figure 3.14(a) that the six physical quantities of Transient 1 can be clearly distinguished in the 2-D t-SNE map, suggesting fundamental intrinsic differences among them. Furthermore, the six quantities at each time step form six line structures that have clear directions as the transient progresses. Such an observation indicates the temporal continuity of these physical quantities during the transient and confirms the necessity to use the LSTM module to learn the temporal information.

The physical quantities of all three transients are plotted together in Figure 3.14(b), from which we can clearly identify six clusters, with each corresponding to a physical quantity. As observed for Transient 1, the physical quantities of Transients 2 and 3 also form lines with directions in



(a)



(b)

Figure 3.14: t-SNE mapping of physical quantity fields at each time step: a) Transient 1 only (arrow direction indicates the progress of transient); b) all three transients.

accordance with the progress of the transient. It can be further found from Figure 3.14(b) that every physical quantity of these three cases has a high similarity at the beginning of the transient, as they are very close to each other within each cluster. This similarity diminishes as the transient progresses, except for V_x and V_y. As observed from the t-SNE map, most physical quantities of the three cases at the end of the transient are much less similar compared to the beginning. The possible reason for the high similarity of V_x and V_y through the whole transient is that both velocity components lack a specific driving source; instead, they are driven by the inlet sodium in the Z direction. As a result, the values of these two fields are relatively small over the whole domain, so the t-SNE map does not distinguish the difference among the three transients.

Such an observation provides an explanation of DCNN-LSTM's performance on Transients 2 and 3: The model has moderate generalization capability for these two transients, as their data have some level of intrinsic similarity with the training data of the model (Transient 1); however, such a similarity is not great enough to fully cover the whole transients, so the model showed a less accurate prediction for Transients 2 and 3 compared to its prediction for Transient 1. Be that as it may, the performance of DCNN-LSTM on Transients 2 and 3 still provides promise that we can leverage a well-trained data-driven model for simulation cases with “extrapolating” inlet and boundary conditions.

4 Uncertainty quantification and model implementation

As risk-informed safety analysis methodology starts to be adopted by nuclear regulatory authorities, uncertainty quantification (UQ) and risk analysis become increasingly important in the nuclear engineering community [23, 24, 25, 26]. In this sense, the modeler should not only provide the prediction of a certain problem, but also provide the uncertainty associated with that prediction. Based on this reason, the UQ of the data-driven DNN model becomes a necessary step before that model can be trusted for engineering applications.

It should be noted that the UQ of DNN prediction is a challenging problem, considering the very large number of parameters (i.e. trainable weights and biases of each neuron) of the network [27]. The Bayesian DNN, a sophisticated method assuming the weights of the DNNs follow certain uncertainty distributions, is considered a state-of-the-art method for the UQ of DNNs. However, the method is too computationally expensive to be used in complicated DNN architectures.

In this work, we utilized two more methods for the UQ of complicated DNNs, i.e. Monte Carlo (MC) dropout and deep ensemble. Both methods are computationally efficient and scalable compared to BNN. We applied these two methods to a densely connected convolutional network, which is developed and trained as coarse-mesh turbulence models for reactor transient analysis. In comparison, the corresponding BNN with the same architecture is also developed and trained. The computational cost and uncertainty evaluation performance of these three UQ methods are comprehensively investigated. With the UQ capability developed for the ML-based closure, we then implemented the trained ML closure into SAM and use it for the thermal stratification modeling. As the first step for the investigation, the above-mentioned work is developed based on a simplified 2-D case, with the ML closure trained with a 2D version of DCNN (termed Dense2D in the following of the report). It should be noted that the developed SAM-ML capability can be naturally extended to 3-D cases, including the DCNN-LSTM model, without significant modifications.

4.1 Monte Carlo dropout

For the neural network without uncertainty estimation, dropout serves as a training technique and can be regarded as a regularization term to mitigate the overfitting issue [28] those neural networks that have dropout layers, the dropout layer will randomly zero some of the elements of the input tensor with probability p using samples from a Bernoulli distribution during the training process. Meanwhile, the outputs are also scaled by a factor of 1/(1-p). MC dropout, on the other hand, serves as an uncertainty estimation method for probabilistic neural network prediction [17]. Such a treatment assumes the output y of a neural network is a conditional probability under the input x, i.e. $p_\theta(y|x)$, where θ stands for the parameters of the neural network. MC dropout only requires training the neural network once. When the baseline model is obtained, MC dropout randomly zeros some of the neurons in the network in each forward pass with probability p from a Bernoulli distribution. Every forward pass in the MC dropout is an independent process, as illustrated in Figure 4.1. The MC dropout can repeat multiple times and generate multiple samples that can be considered as a Bayesian approximation of $p_\theta(y|x)$.

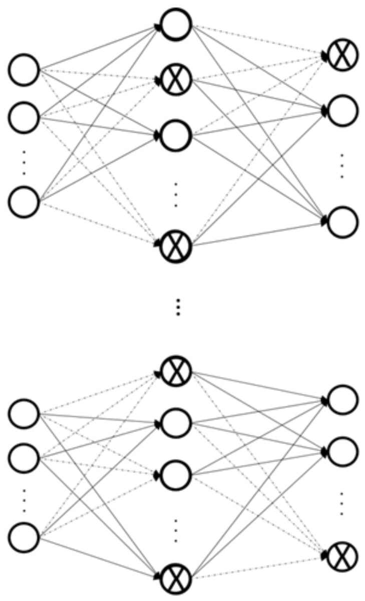


Figure 4.1: MC dropout samples based on one single baseline model.

4.2 deep ensemble

Similar to MC dropout, deep ensemble also involves generating multiple samples to form an ensemble and evaluate the statistics of $p_\theta\left(y|x\right)$ for the UQ purpose [29]. On the other hand, deep ensemble requires training the one neural network multiple times. When using deep ensemble, the network Dense2D outputs two scalar field in the final layer, corresponding to the predictive mean $\mu(\boldsymbol{x})$ and variance $\sigma^2(\boldsymbol{x})$ of turbulent viscosity. In this sense, the RMSE cannot be used as the loss function for training since it cannot reflect the predictive uncertainty. Instead, the negative log-likelihood function is chosen as the loss function for training the neural network. By assuming the training data follows Gaussian distributions, the negative log-likelihood function can be expressed as:

$$-\log p_\theta\left(\hat{y}_n|x_n\right) = \frac{\log \sigma_\theta^2\left(x_n\right)}{2} + \frac{\left(\hat{y}_n - y_n\right)^2}{2\sigma_\theta^2\left(x_n\right)} \ . \tag{19}$$

Deep ensemble usually relies on a randomization-based ensemble generation approach. In this practice, we randomly initialize the parameters of neural network and randomly shuffle the data samples in each training. After M samples are generated, the mean and variance of the ensemble can be calculated as:

$$\begin{aligned} \hat{\mu}\left(\boldsymbol{x}\right) &= \tfrac{1}{M}\textstyle\sum_{i=1}^{M} \hat{\mu}_{\theta_i}\left(\boldsymbol{x}\right) \ , \\ \sigma^2(\boldsymbol{x}) &= \tfrac{1}{M}\textstyle\sum_{i=1}^{M}\left(\hat{\sigma}_{\theta_i}^2\left(\boldsymbol{x}\right) + \hat{\mu}_{\theta_i}^2\left(\boldsymbol{x}\right)\right) - \hat{\mu}^2\left(\boldsymbol{x}\right). \end{aligned} \tag{20}$$

The model prediction then follows the Gaussian distribution as

$$\hat{y}\left(\boldsymbol{x}\right) \backsim \mathcal{N}(\hat{\mu}\left(\boldsymbol{x}\right), \sigma^2(\boldsymbol{x})). \tag{21}$$

Although deep ensemble requires multiple independent training of the neural network, its advantage is that the training is highly parallelizable, thus ensuring the scalability.

4.3 Bayesian neural network

Both MC dropout and deep ensemble are based on deterministic neural network: MC dropout introduces uncertainty through randomly zeros certain neurons in the deterministic network; while deep ensemble introduces uncertainty through training a deterministic neural network multiple times with randomly weight initialization and training data shuffling. Unlike these two methods, BNN treats the parameter θ as random variables, making it a real probabilistic neural network in nature. The difference of these two types of neural networks is illustrated in Figure 4.2.

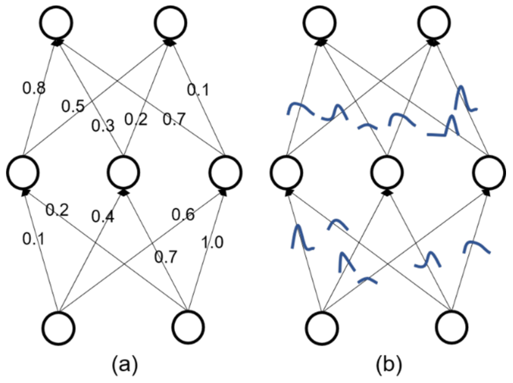


Figure 4.2: (a) deterministic neural network with each weight has a fixed value; (b) probabilistic neural network with each weight follows a certain uncertainty distribution.

The training of BNN is a more challenging task compared to training a deterministic neural network with classical backpropagation [30]. In this work, we sample the weights w and biases b of the BNN with the following equation:

$$\begin{aligned} w &= \mathcal{N}(0,1) \times \log\left(1 + \rho\right) + \mu, \\ b &= \mathcal{N}(0,1) \times \log\left(1 + \rho\right) + \mu. \end{aligned} \tag{22}$$

where ρ parameterizes the standard deviation and μ parametrizes the mean for the samples. For BNN, both ρ and μ are trainable parameters for each neuron. The loss function with regard to w and b can be defined based on the Kullback-Leibler Divergence:

$$L[y,(w,b)] = KL[q(w,b|z)||P(w,b)] - \mathbb{E}_{q(W,b|z)}[logP(y|(W,b))], \tag{23}$$

where z is a latent variable (ρ, μ) that defines the weight and bias distribution, $P(w, b)$ is the prior distribution of the weight and biases, $q(w, b|z)$ is the posterior distribution of the weight and biases, given the latent variable z. We use the variational inference method to optimize and reduce the loss function of Eq.23.

Eq.23 requires multiple samples to estimate its uncertainty terms. However, unlike deep ensemble, such multiple samples are very difficult to run parallel, making the scalability the most limiting issue for applying BNN for complex neural network.

4.4 UQ results

The UQ for DNN with the above-mentioned three methods are studied. Both the performance and computational cost of the three methods are evaluated. To reflect the uncertainty nature of the problem, we impose a Gaussian noise to the training data:

$$\begin{gathered} y_{train} = y_{CFD} + \ \varepsilon\ , \\ \varepsilon \sim \mathcal{N}(0,\ 0.1 \times y_{CFD}). \end{gathered} \tag{24}$$

Such a treatment means we assume the obtained CFD data has a 10% uncertainty with regard to its value.

For MC dropout, the Dense2D baseline model is the only neural network that needed. We perform 20 forward passes of the baseline Dense2D. In each pass, we randomly zero certain neurons after every convolutional layer with a probability of 0.15 following a Bernoulli distribution. The obtained mean and two standard deviation bound from MC dropout, in comparison to the CFD data with uncertainty, is depicted in Figure 4.3. The quantitative uncertainty evaluation at y=0 (vertical line) and z=0 (horizontal line) is depicted in Figure 4.4.

It can be found from Figures 4.3 and 4.4 that the mean of MC dropout is in good agreement with the CFD results, and its predicted uncertainty bound is on the same scale as the CFD data uncertainty. However, a visual look at Figure 4.3 indicates a non-smooth and oscillating uncertainty filed predicted by MC dropout. Such an uncertainty pattern suggests that the randomly dropout neurons in a complicated neural network may not generate smooth uncertainty prediction.

In the baseline Dense2D model, there is only one output: the turbulent viscosity field. For deep ensemble, we keep the majority of the Dense2D architecture and only increased the final output channel from 1 to 2: one represents the mean of the turbulent viscosity field, the other represents the variance of the field. In this practice, we trained a total of 20 modified Dense2D model, with their weights initialized randomly. We perform the training on Argonne's LCRC Blues cluster, and used 5 GPU nodes for the training. Each node has 2 NVIDIA Tesla K40m GPUs. Since deep ensemble is highly parallelizable, we can train cases simultaneously on the 5 nodes. The computational cost is recorded. By assuming the network prediction follows a Gaussian process, the mean and 2 standard deviation bounds can be calculated following Eq. 20, and are depicted in Figure 4.5. Similar to MC dropout, the quantitative uncertainty evaluation at y=0 (vertical line) and z=0 (horizontal line) is

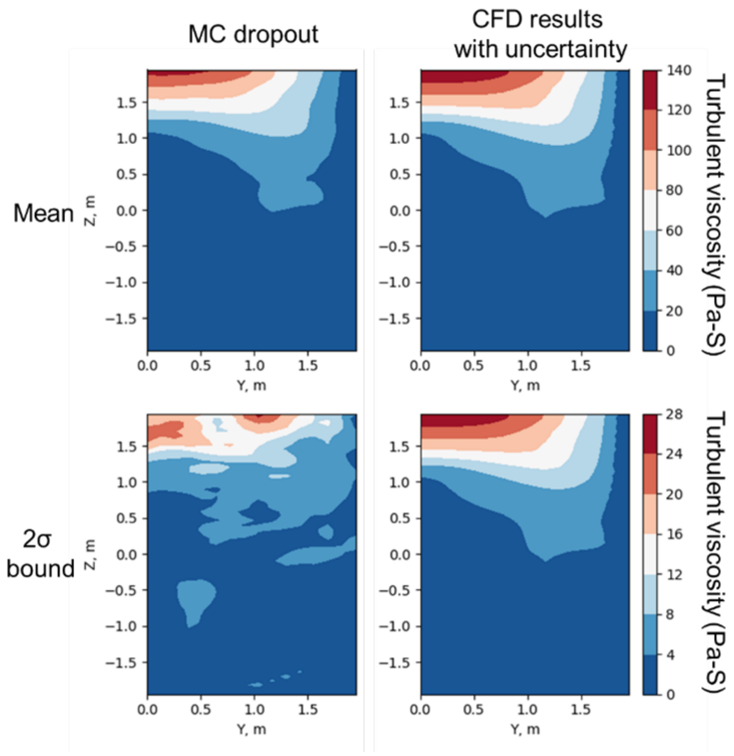


Figure 4.3: Mean and 2σ bound of MC dropout prediction, in comparison with CFD results (at t = 20.11 s).

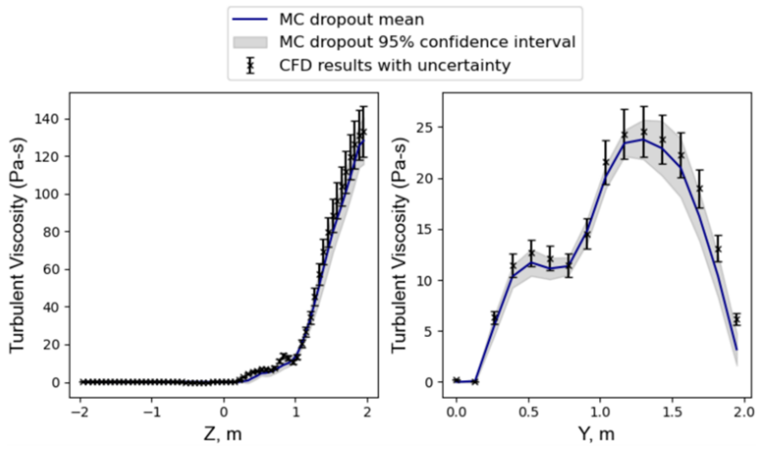


Figure 4.4: Mean and 95% confidence interval of MC dropout predictions, in comparison with CFD results (at t = 20.11 s).

depicted in Figure 4.6.

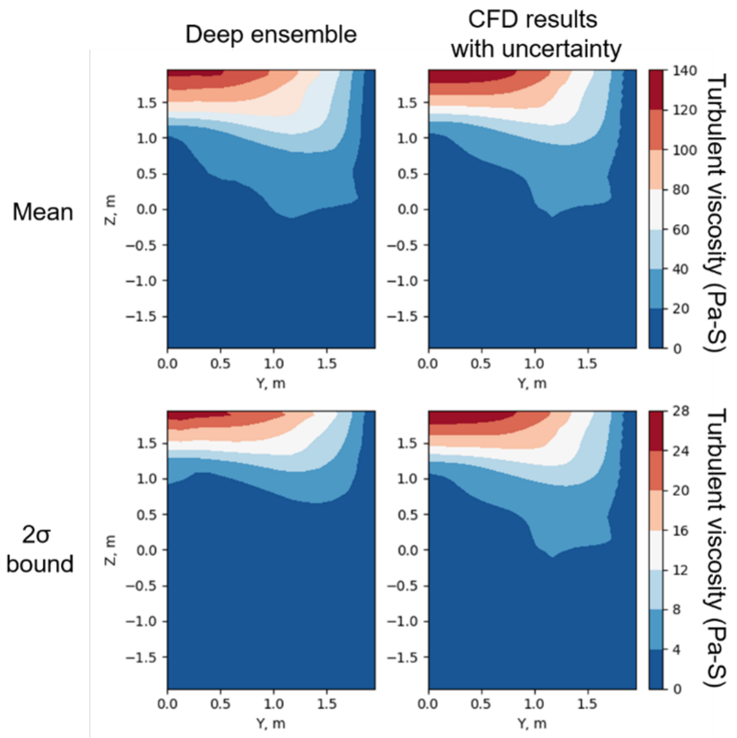


Figure 4.5: Mean and 2σ bound of deep ensemble prediction, in comparison with CFD results (at t = 20.11 s).

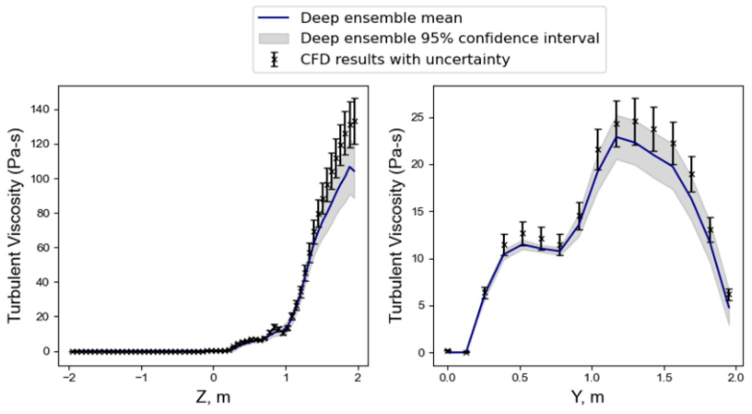


Figure 4.6: Mean and 95% confidence interval of deep ensemble predictions, in comparison with CFD results (at t = 20.11 s).

It can be found from Figure 4.5 that, in contrast to MC dropout, deep ensemble can generate smooth uncertainty prediction that is consistent with the CFD uncertainty. In the meantime, the predicted mean is in good agreement with the CFD results. The quantitative comparison shown in Figure 4.6 also confirmed the good agreement. Such observation indicates deep ensemble is an appropriate tool for the UQ of Dense2D, as for most engineering problems, it is safe to assume its prediction follows Gaussian distribution.

For BNN, the Dense2D baseline model has been transformed to its Bayesian version, with all its parameters (weights and biases) follows a certainty distribution and are sampled following Eq. 22.

When training the BNN version of Dense2D, we choose to sample the weights and biases 20 times based on which the statistics of the distribution and the loss function Eq. 23 are calculated. That is similar to training a deterministic neural network model 20 times, but it is run in parallel. In this work, we use one GPU node for the training. The obtained results are depicted in Figures 4.7 and 4.6. It can be found that the mean of BNN prediction agrees with the CFD results well, the 2 standard deviation uncertainty bound is also on the same scale as the CFD uncertainty. On the other hand, it overestimated the uncertainty range in the lower part of the field. The quantitative comparison is shown in Figure 4.8 also indicates the BNN prediction missed a saddle point in the horizontal line (z = 0 m).

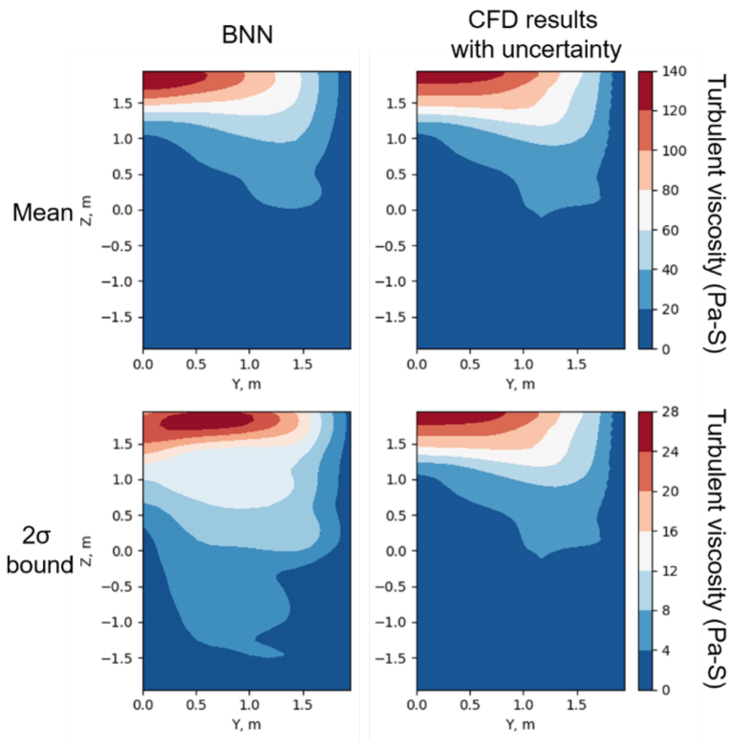


Figure 4.7: Mean and 2σ bound of deep ensemble prediction, in comparison with CFD results (at t = 20.11 s).

The computational costs of these three methods are evaluated, and denoted by GPU node hours (as they are training on the same cluster) and actual time, as summarized in Table 4.1. The MC dropout has the lowest computer cost as it only requires to train the model once. Deep ensemble, on the other hand, has the second-lowest computational cost among these three methods, but its actual time cost is not high. This is because the good scalability feature makes it can be trained on different nodes simultaneously and independently. BNN has the highest computational cost and as

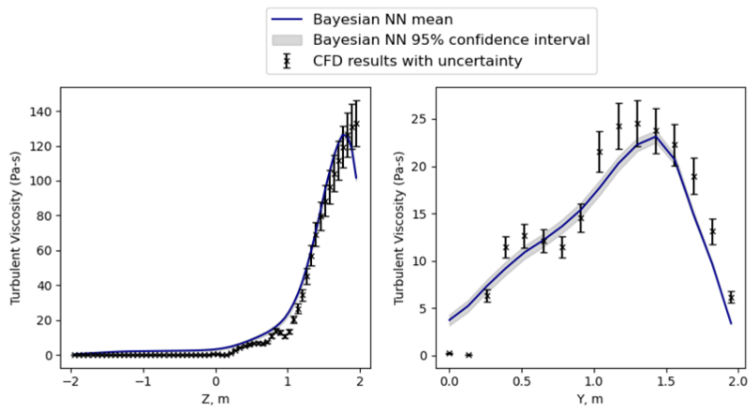


Figure 4.8: Mean and 95% confidence interval of deep ensemble predictions, in comparison with CFD results (at t = 20.11 s).

Table 4.1: Comparison of the computational cost of three UQ methods

	GPU node hours	Actual hours
MC dropout	6	6
Deep ensemble	80	16
BNN	110	110

it cannot be trained in parallel, it's training also costs the highest actual time. Such an observation meets with our prior anticipation that BNN is a computational expensive tool for complex neural networks such as the Dense2D model studied in this work.

The observation suggests that the deep ensemble can achieve good prediction and uncertainty estimation for the investigated problem, and the actual computational cost is significantly less compared to the Bayesian neural network. Based on this reason, deep ensemble could be used as an efficient UQ method for complex neural network architectures that developed for scientific modeling problems. On the other hand, deep ensemble assumes the neural network prediction follows the Gaussian distribution. Such an assumption, although often valid for engineering problems, still need attention for its validity before applying the method to a certain problem.

4.5 Implementing deep ensemble model into SAM

With a properly trained Dense2D deep ensemble model developed, we can implement it into SAM's multi-D module to integrate the data-driven closure into the physics-based solving process, i.e. the proposed SAM-ML approach. We developed a flexible data exchange interface between SAM and the Dense2D model, based on which the integration becomes possible. The interface reads in the variable value of each element in the computation domain and organizes it into a scalar field based on its coordinate. After the Dense2D forward evaluation, the interface maps the PyTorch tensor value back to the corresponding element based on its coordinate value. In the interface, we also incorporate flexible mapping between the computation domain mesh and the Dense2D input/output tensor through interpolation. Such a process ensures we can have a flexible simulation mesh setup that is not constrained by the dimensionality of the Dense2D input/output. The data exchange process is depicted in Figure 4.9.

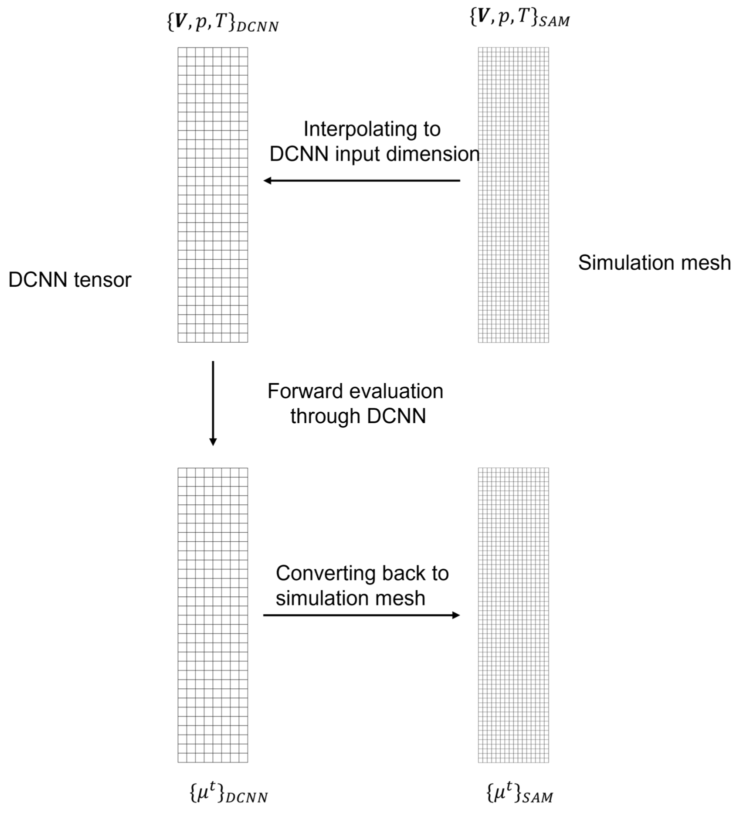

$\{\boldsymbol{V}, p, T\}_{DCNN}$
$\{\boldsymbol{V}, p, T\}_{SAM}$
Interpolating to DCNN input dimension
DCNN tensor
Simulation mesh
Forward evaluation through DCNN
Converting back to simulation mesh
$\{\mu^t\}_{DCNN}$
$\{\mu^t\}_{SAM}$

Figure 4.9: Flexible conversion between SAM and Dense2D.

The developed SAM-ML can then be used to run simulations of reactor transients that involve thermal stratification in large open volumes. As we integrated the trained data-driven model into the SAM, the evaluation of Dense2D is coupled with the physics-solving process in SAM's multi-D module. The whole solving process of SAM-ML is depicted in Figure 4.10. The Dense2D-based turbulence model requires only one forward evaluation of the Dense2D model to obtain the turbulent viscosity field μ^t in one iteration. We found that the running time of the SAM-ML simulation does not have significantly more computation time than the zero-equation turbulence model setup—an observation that confirms the computational efficiency of the developed SAM-ML approach.

To be consistent with the CFD simulation, the coarse-mesh 2D SAM-ML simulation is also based on the geometry of ABTR conceptual design, as depicted in Figure 4.11. The simulation domain is a 2-D axisymmetric based on cylindrical coordinate. The inlet is in the left bottom area, while the outlet is in the right center area. The left side of the boundary is axis, while the rest of the boundaries are set as walls. The computation domain is decomposed into two blocks: The turbulence in the major block representing the hot pool is modeled with Dense2D, while the turbulence in the extruded block, designed to avoid reverse flow, is modeled with zero-equation mixing length model. Unlike the 3-D CFD simulation that has more than 1.1 million cells, this

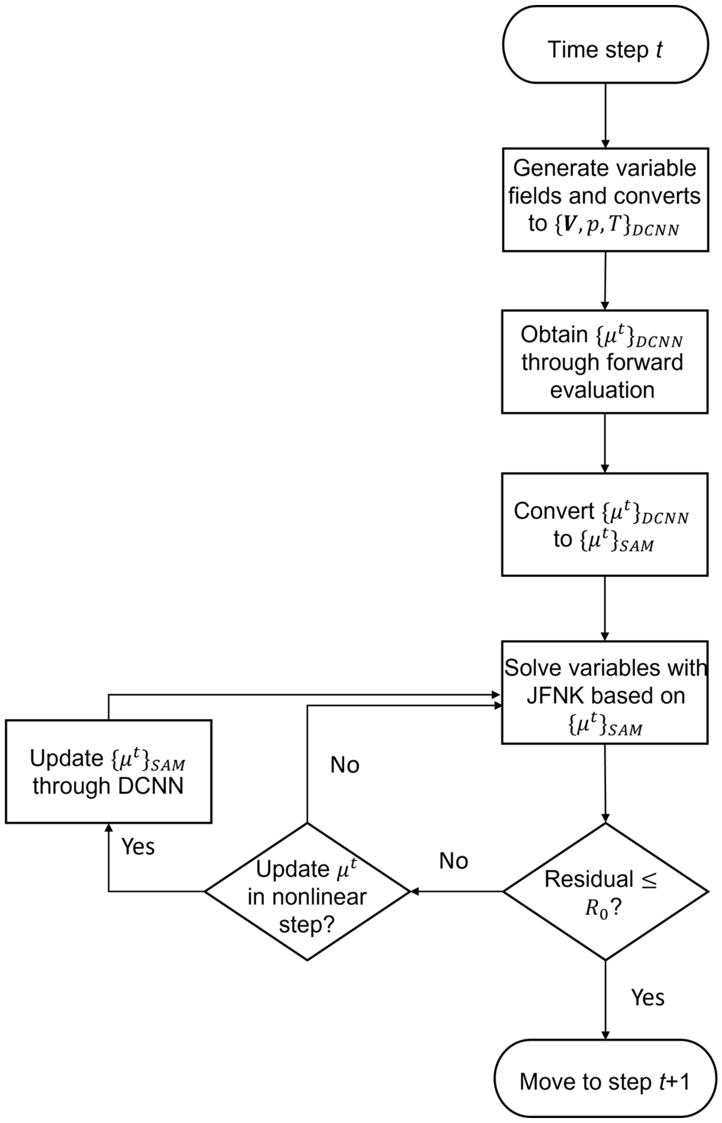

Time step t
Generate variable fields and converts to $\{\boldsymbol{V}, p, T\}_{DCNN}$
Obtain $\{\mu^t\}_{DCNN}$ through forward evaluation
Convert $\{\mu^t\}_{DCNN}$ to $\{\mu^t\}_{SAM}$
Solve variables with JFNK based on $\{\mu^t\}_{SAM}$
Residual ≤ R_0?
No
Update μ^t in nonlinear step?
No
Yes
Update $\{\mu^t\}_{SAM}$ through DCNN
Yes
Move to step t+1

Figure 4.10: Solving process of SAM-ML based on JFNK.

coarse-mesh SAM-ML simulation contains only 8272 elements.

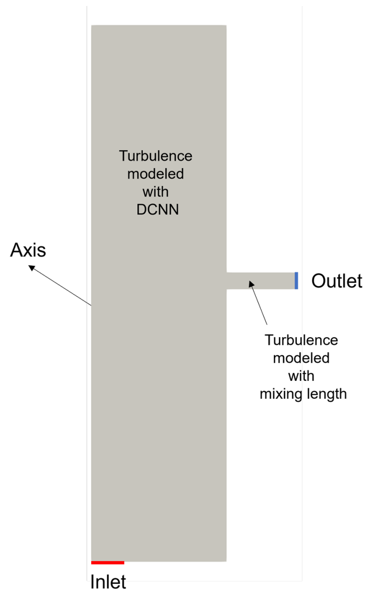

Turbulence modeled with DCNN
Axis
Outlet
Turbulence modeled with mixing length
Inlet

Figure 4.11: Simplified 2-D SAM-ML simulation.

The SAM-ML simulation is also based on a loss-of-flow transient, the inlet conditions are computed based on SAM's 1D simulation with a given reactor power and pump head. The obtained inlet mass flow rate and the temperature of the liquid sodium are depicted in Figure 4.12. In the transient, the input mass flow rate decreased from 632 kg/s to ¡50 kg/s in less than 200 seconds, while the temperature first decreased, due to the power drop, then increased due to the low mass flow rate and non-negligible decay heat. Such a transient involves very typical thermal stratification phenomena. Direct comparisons between the fine-mesh CFD, the coarse-mesh SAM-ML, and the coarse-mesh SAM zero-equation at t = 100s and 250s are depicted in Figure 4.13 and Figure 4.14, respectively.

We observed that the SAM-ML simulation converges well in each time step with the JFNK method, and the simulation speed is also similar to the simulation with only a zero-equation turbulence model. This confirms the good compatibility between the ML model and the SAM, as well as the computational efficiency of integrating the Dense2D model with SAM's multi-D module. Direct comparisons between the fine-mesh CFD simulation and the coarse-mesh SAM-ML at $t =$ 100s and 250s are depicted in Figure 4.13 and Figure 4.14, respectively.

The figures show that the simulation results with SAM zero-equation model have large discrepancies with the original CFD results, while also exhibiting a very unstable pattern. The reason for such a behavior is the mixing length model significantly underestimated the turbulent viscosity, resulting unphysical oscillation in the velocity and temperature fields. Such an observation reveals the limitation of the zero-equation turbulence model, especially when the mixing length value is not tuned.

On the other hand, although discrepancies can be observed, the general trend of the turbulent

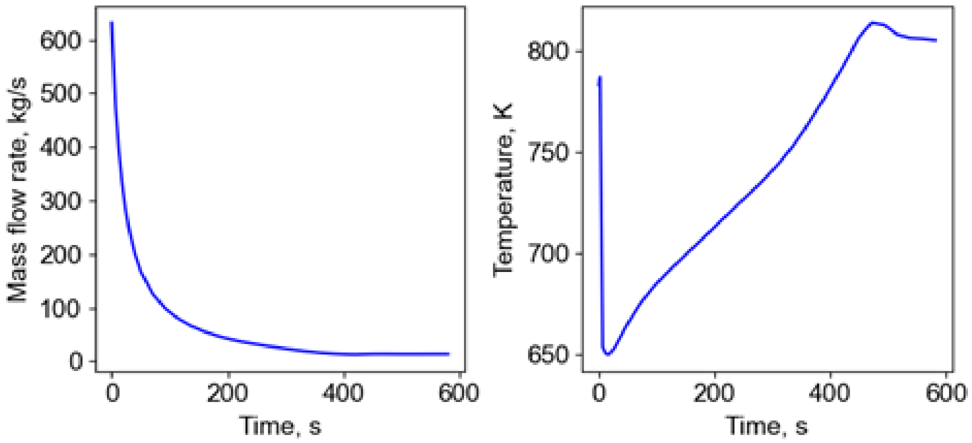


Figure 4.12: Inlet conditions of mass flow rate and temperature.

viscosity predicted in the SAM-ML is consistent with the CFD results, indicating that the trained DCNN model with deep ensemble is robust and has good generalization capability. In addition, with the turbulent viscosity captured by the SAM-ML with reasonable accuracy, the major variables, including velocity components, temperature, and pressure, solved in SAM-ML also showed good agreement with the CFD results. The existing discrepancy can be explained by a few factors: the intrinsic numerical difference between STAR-CCM+ (finite volume based) and MOOSE (finite element based), the non-negligible geometrical distortion resulting from converting a 3-D simulation to 2-D, as well as the numerical diffusion effect in SAM-ML due to the coarse mesh setup.

A quantitative comparison between the SAM-ML results and the CFD simulation results of the liquid sodium outlet temperature, a key safety parameter in the thermal stratification problem, is shown in Figure 4.15. In the figure, we observe good agreement between the results of SAM-ML and CFD. The largest discrepancy occurs at the beginning of the transient, where a fluctuation is observed in the SAM-ML simulation, while the CFD results remained stable. After the thermal stratification is formed after t 50 s, the discrepancy between SAM-ML and CFD falls below 10 K. The results of SAM zero-equation, on the other hand, showed strong oscillation pattern due to the underestimated turbulent viscosity, while also significantly overestimated the sodium outlet temperature. Such an observation further confirms the validity of the SAM-ML approach.

With the SAM-ML capability developed, we can train a machine learning model with a larger dataset that can cover many transients with different geometry setups. As a result, the predictive capability of SAM-ML for thermal stratification in large open volumes can be significantly improved. The developed SAM-ML approach also has the potential to be extended for a variety of simulation problems that require closure relations.

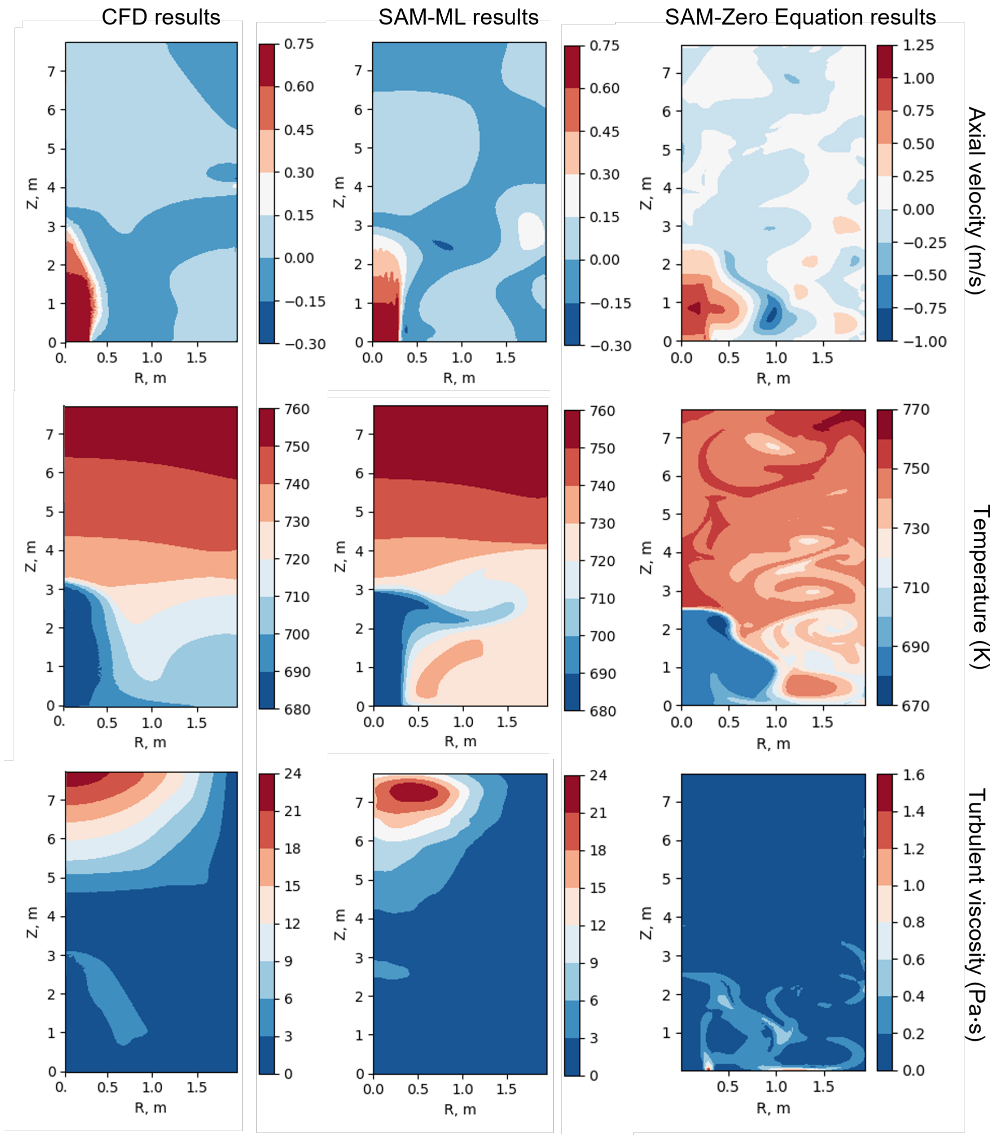


Figure 4.13: Comparison between CFD and SAM-ML simulation results ($t = 100.0$ s).

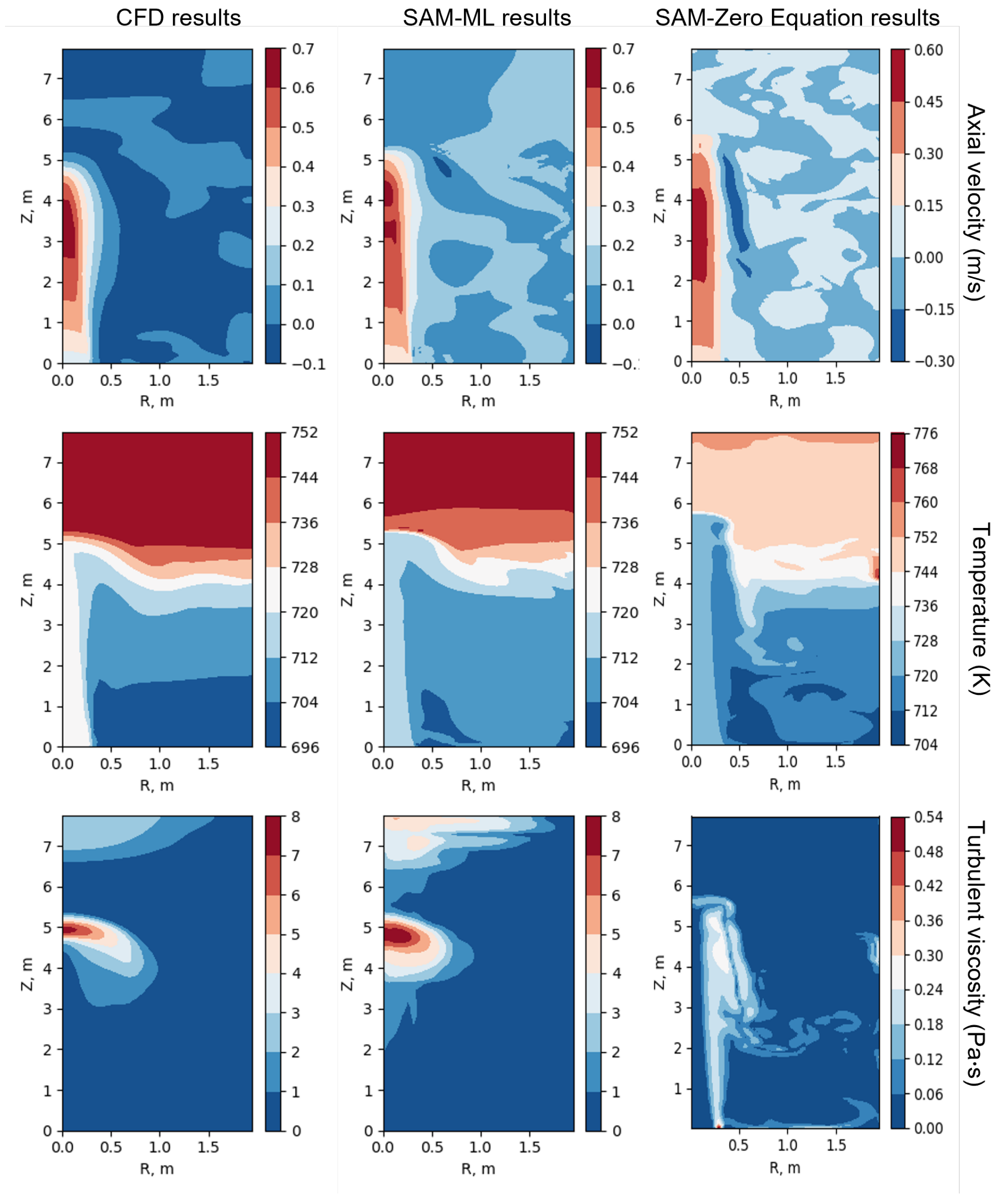


Figure 4.14: Comparison between CFD and SAM-ML simulation results (t = 250.0 s).

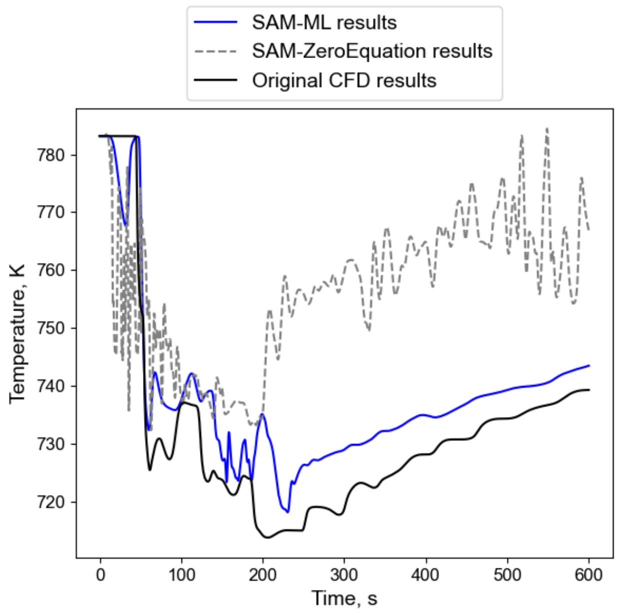


Figure 4.15: Inlet conditions of mass flow rate and temperature.

5 Model improvement with higher fidelity data

It should be noted that the RANS-based CFD simulation has limitation on capturing the turbulence effect on complex thermal-fluid flows, such as the turbulent-laminar flow transition. Considering this, the DNN models trained with RANS data can be further improved with the support of higher-fidelity data. Based on this reason, we performed additional simulation using the large-eddy simulation (LES) for a few snapshots of the transient. The obtained data can be used for further improvement of the DNN model in a future work.

5.1 Large-eddy simulation

Governing equations

The maximum and minimum temperatures of the inlet flow in the first $20\,s$ are $787.07\,K$ ($\rho = 829.02\,kg/m^3$) and $649.77\,K$ ($\rho = 861.74\,kg/m^3$), respectively. The density variation of these two temperature values is approximately 4% relative to the one of minimum temperature. The error may be more substantial if the Boussinesq approximation is used to model the buoyancy effects. Therefore, the `buoyantPimpleFoam` solver in OpenFOAM [31] is employed to perform the large-eddy simulations (LES) in the present study. The filtered governing equations are given by:

$$\frac{\partial \rho}{\partial t} + \nabla \cdot \left(\rho \overline{\mathbf{U}}\right) = 0, \tag{25a}$$

$$\frac{\partial}{\partial t}\left(\rho \overline{\mathbf{U}}\right) + \nabla \cdot \left(\rho \overline{\mathbf{U}} \otimes \overline{\mathbf{U}}\right) = -\nabla \overline{p} + \nabla \cdot \left(\mu_{eff}\left(2\mathbf{S} - \frac{2}{3}\mu\left(\nabla \cdot \overline{\mathbf{U}}\right)\mathbf{I}\right)\right) + \nabla \cdot \tau + \rho \mathbf{g}, \tag{25b}$$

$$\frac{\partial}{\partial t}\left(\rho h\right) + \nabla \cdot \left(\rho \overline{\mathbf{U}} h\right) + \frac{\partial}{\partial t}\left(\rho K\right) + \nabla \cdot \left(\rho \overline{\mathbf{U}} K\right) - \frac{\partial \overline{p}}{\partial t} = \nabla \cdot \left(\alpha_{eff} \nabla h\right) + \rho \overline{\mathbf{U}} \cdot \mathbf{g}, \tag{25c}$$

where ρ is the density, t the time, $\overline{\mathbf{U}}$ the resolved velocity, $\overline{p}$ the resolved pressure, μ_{eff} the effective viscosity, $\mathbf{S}\left(= \frac{1}{2}\left(\nabla \overline{\mathbf{U}} + \left(\nabla \overline{\mathbf{U}}\right)^T\right)\right)$ the strain rate tensor of the resolved field, $\mathbf{I}$ the unit tensor, $\mathbf{g}$ the acceleration due to gravity, $\tau\left(= -\rho\left(\overline{\mathbf{U} \otimes \mathbf{U}} - \overline{\mathbf{U}} \otimes \overline{\mathbf{U}}\right)\right)$ the subgrid-scale (SGS) stress tensor, h the enthalpy, K the specific kinetic energy, and α_{eff} the summation of laminar and turbulent thermal diffusivities. It should be noted that the `buoyantPimpleFoam` solver does not employ the Boussinesq approximation, and it's governing equations (Eq. (25)) are in the form of compressible flows. The state equation is stated in Eq. (9a).

The Smagorinsky model [32] is employed to approximate the SGS stress tensor τ, which is given by:

$$\tau = \frac{1}{3}\mathbf{tr}\left(\tau\right)\mathbf{I} + \left(\tau - \frac{1}{3}\mathbf{tr}\left(\tau\right)\mathbf{I}\right) \approx \frac{1}{3}\mathbf{tr}\left(\tau\right)\mathbf{I} + 2\mu_t \mathbf{S} = -\frac{2}{3}\rho k \mathbf{I} + 2\mu_t \mathbf{S}, \tag{26}$$

where k is the SGS kinetic energy and μ_t is the eddy viscosity, which is evaluated as:

$$\mu_t = \rho C_k \Delta \sqrt{k}, \tag{27}$$

where C_k is a constant coefficient with a default value of 0.094 and Δ the subgrid length scale which is solved as the cubic root of the cell volumes in the present study. The SGS kinetic energy k is the solution of the quadratic equation $\frac{C_e}{\Delta}k^2 + \frac{2}{3}\mathbf{tr}\,(\mathbf{S})k + 2C_k\Delta\,(\mathbf{dev}\,(\mathbf{S}) : \mathbf{S}) = 0$, where the default value of C_e is 1.048.

Turbulent inlet and initial conditions

The difficulties and challenges of the LES simulations in the present study are generating fully developed turbulent inlet and initial conditions [33]. Various methods can be used to generate turbulent inlet velocities, *i.e.*, precursor simulations, recycling method and synthetic turbulence generations. Detail descriptions and comparisons of these methods are available in [34].

In the present study, the mean values of inlet velocity are time dependent and identical with the RANS simulations. The inlet condition is generated by extending the inlet patch in the upstream direction and running a precursor LES simulation in the extended pipe (see Figure 5.1) with recycling method. Then the turbulent flow will be interpolated into the inlet patch of the realistic domain.

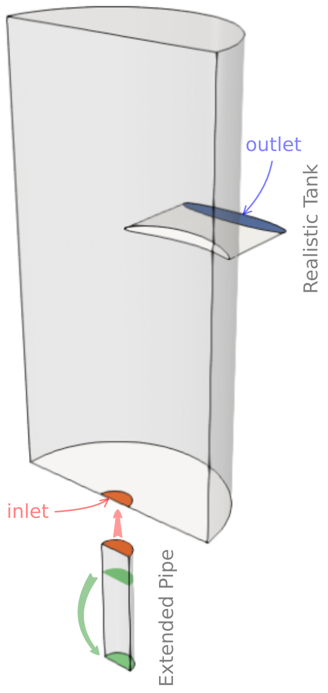
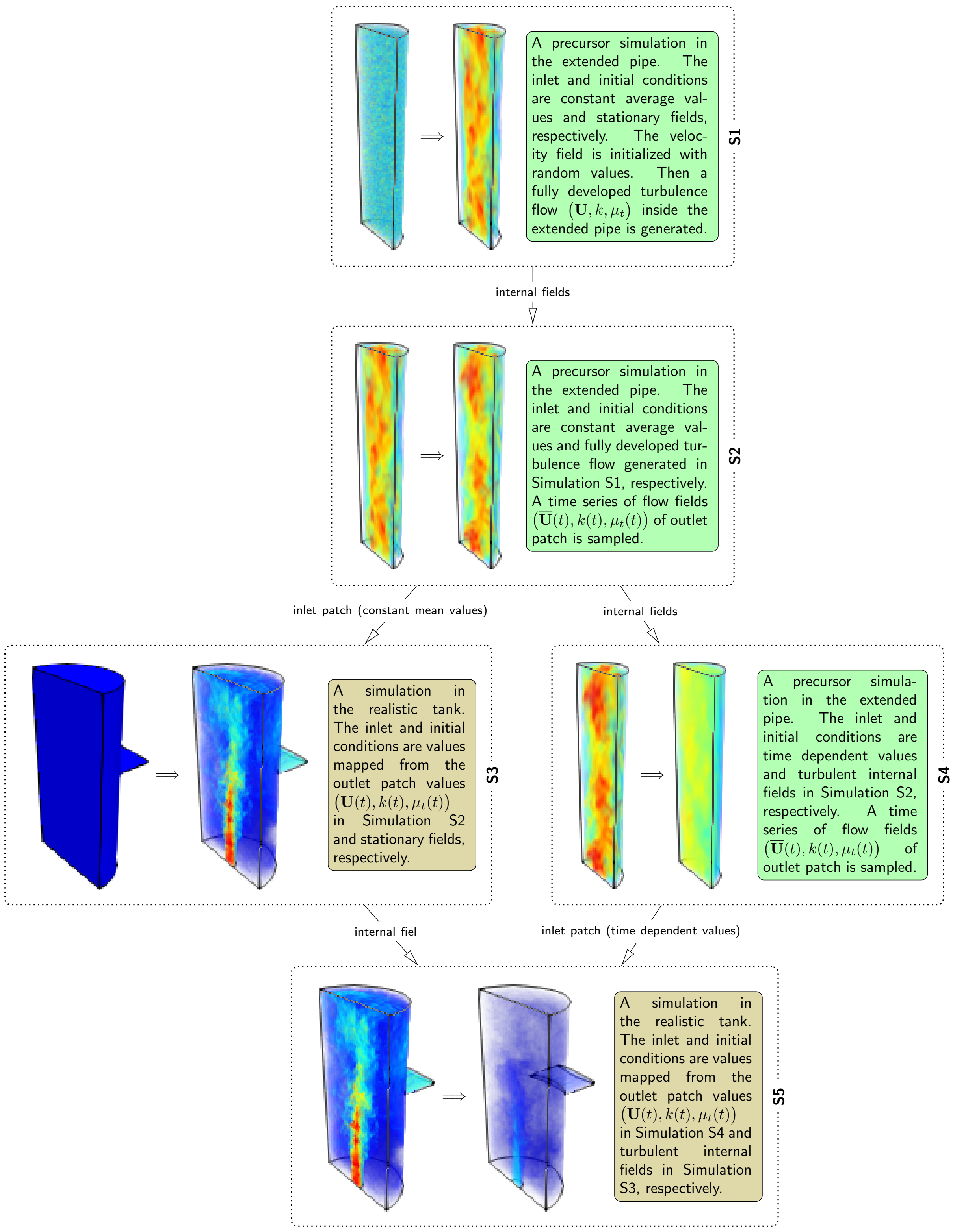


Figure 5.1: Extended pipe and realistic tank.

The detail procedures are shown in Figure 5.2. In Simulation S1, the inlet and initial conditions are constant average values and stationary fields, respectively. The velocity field is initialized with random values to accelerate the transition from laminar to turbulent flows. Then a fully developed turbulence flow $\left(\overline{\mathbf{U}}, k, \mu_t\right)$ inside the extended pipe is generated. In Simulation S2, the inlet and initial conditions are constant average values and fully developed turbulence flow generated in Simulation S1, respectively. A time series of flow fields $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ of outlet patch is sampled. In Simulation S3, the inlet and initial conditions are values mapped from the outlet patch values $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ in Simulation S2 and stationary fields, respectively. In Simulation S4, the inlet and initial conditions are time dependent values and turbulent internal fields in Simulation S2, respectively. A time series of flow fields $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ of outlet patch is sampled. Finally, in Simulation S5, the inlet and initial conditions are values mapped from the outlet patch values $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ in Simulation S4 and turbulent internal fields in Simulation S3, respectively.


S1
internal fields
S2
inlet patch (constant mean values)
internal fields
S3
S4
internal fiel
inlet patch (time dependent values)
S5

A precursor simulation in the extended pipe. The inlet and initial conditions are constant average values and stationary fields, respectively. The velocity field is initialized with random values. Then a fully developed turbulence flow $\left(\overline{\mathbf{U}}, k, \mu_t\right)$ inside the extended pipe is generated.

A precursor simulation in the extended pipe. The inlet and initial conditions are constant average values and fully developed turbulence flow generated in Simulation S1, respectively. A time series of flow fields $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ of outlet patch is sampled.

A simulation in the realistic tank. The inlet and initial conditions are values mapped from the outlet patch values $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ in Simulation S2 and stationary fields, respectively.

A precursor simulation in the extended pipe. The inlet and initial conditions are time dependent values and turbulent internal fields in Simulation S2, respectively. A time series of flow fields $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ of outlet patch is sampled.

A simulation in the realistic tank. The inlet and initial conditions are values mapped from the outlet patch values $\left(\overline{\mathbf{U}}(t), k(t), \mu_t(t)\right)$ in Simulation S4 and turbulent internal fields in Simulation S3, respectively.

Figure 5.2: Overall procedures of the LES simulations.

LES simulation results

The computational mesh of the extended pipe (see Figure 5.3) is generated by using the `Trimmer` mesher in STARCCM+. The mesh is uniform in the majority part of the domain. The maximum size of the cells is $1.55\,cm$, and the elements near inlet patch curve have been refined up to 1/4 of the maximum value. 10 prismatic cell layers with a total thickness of $3.1\,cm$ are created next to the wall boundaries. The overall cell numbers of the extended pipe and tank are 161476 and 14903603, respectively. A close view of the mesh close to the inlet patch is shown in Figure 5.3.

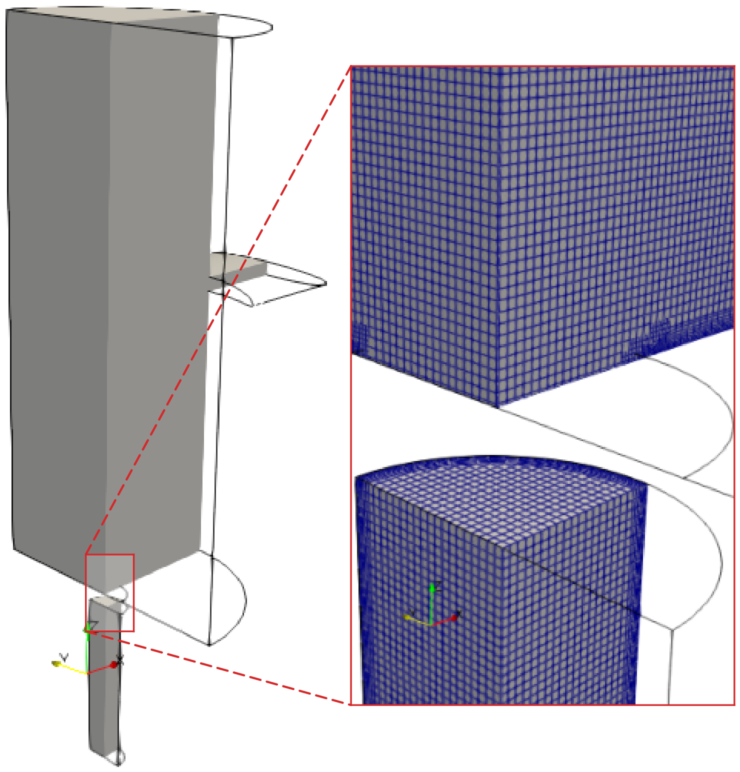


Figure 5.3: A close view of the mesh close to the inlet patch.

In the present study, the unsteady, convection and diffusion terms are discretized by the second-order Euler, bounded second-order upwind and central-differencing schemes, respectively. The velocity-pressure coupling is achieved through the Pressure Implicit Split Operator (PISO) algorithm [35]. The convergence residual of the velocity and pressure fields is set to 1.0×10^{-3} in each outer iteration. The time step value is controlled by the Courant number Co (also referred as CFL number). The maximum Co is 1.0 in the present study.

The time history of the volume-averaged ratio between the resolved and total turbulent kinetic energies in Simulation S1 is shown in Figure 5.4. This mesh resolved approximate 85.6% of the total turbulent kinetic energy. Therefore the meshing methods and sizes in the extended pipe are also employed in the full size tank.

The Simulation S5 is running up to a time of 30 s. The inlet velocity magnitude $|\overline{\mathbf{U}}|$ and eddy

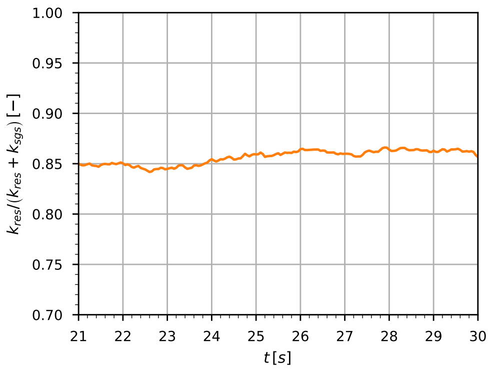
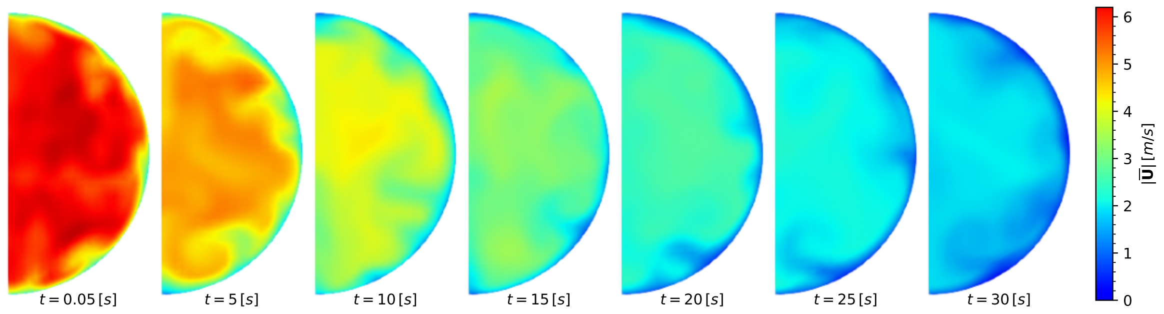
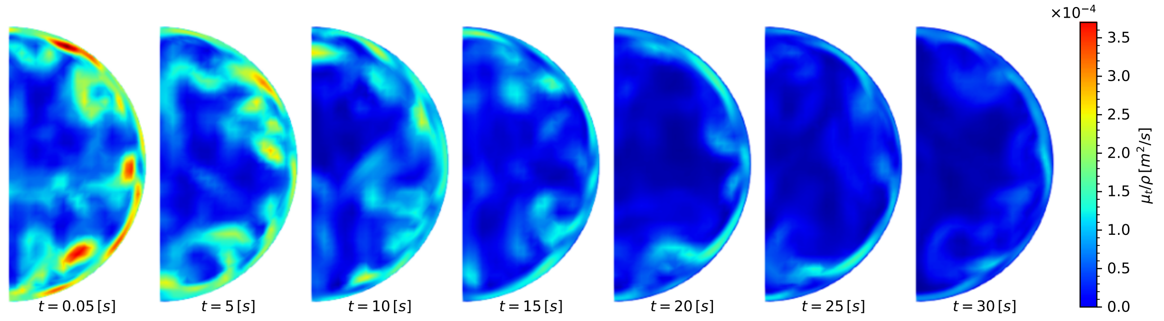


Figure 5.4: Time history of the volume-averaged ratio between the resolved and total turbulent kinetic energies.

kinematic viscosity $\nu_t(=\mu_t/\rho)$ (from Simulation S4) at different time instants are shown in Figure 5.5. As expected, time dependent turbulence inlet conditions have been generated.



(a) Inlet velocity



(b) Inlet eddy kinematic viscosity

Figure 5.5: Inlet velocity and eddy kinematic viscosity at different time instants in Simulation S5.

Figure 5.6 shows the internal velocity fields at different time instants. The initial velocity field is mapped from Simulation S3, which has the maximum values in the simulation. then the inlet velocity is decreasing following the time dependent mean value profile.

The results shows the unsteady behaviors of jet flow interacting with the internal flow of the

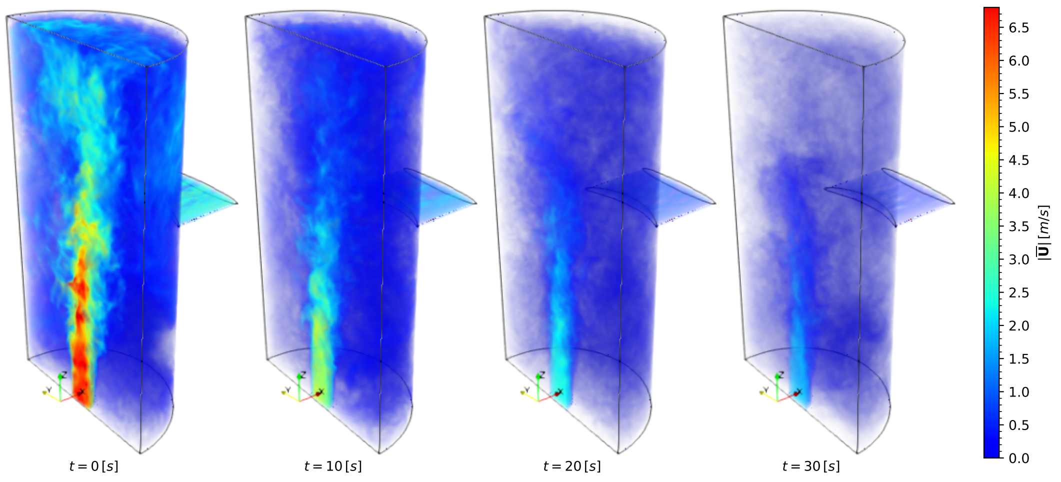


Figure 5.6: Internal velocity fields at different time instants in Simulation S5.

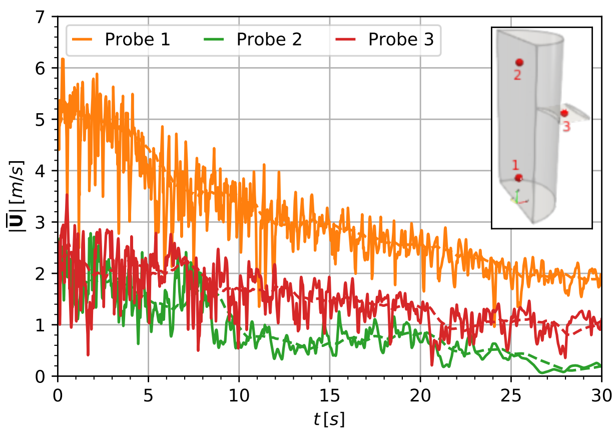
tank. The potential core length is approximate half of the tank height initially, and then decreases as the average inlet velocity reduces from 5.16 m/s to 1.95 m/s. Compared with the URANS result, the LES ones provide more detail information of the internal flows, which will give better corrections to the ML models.

Three probes are set to monitor the flow velocities. As shown in Figure 5.7a, the Probe 1 is placed inside of the initial potential core region of the jet flow (see Figure 5.6). The Probes 2 and 3 are put close to the top surface and outlet patch of the tank, respectively. Figure 5.7a shows the time history of velocity magnitudes at different probes. The solid and dashed lines represent the instantaneous and average values, respectively. The velocity values in Probe 1, which is close to the inlet patch, are higher than the other two probes. In the early stage, the velocities at Probes 2 and 3 are close, and then the Probe 2 values reduce more then the ones at Probe 3. The turbulence energy spectra of the three probes are shown in Figure 5.7b. The inertial (which is specified by the $-5/3$ slope) and dissipation (for higher frequencies) sub-ranges indicate that the simulation has resolved the inertial sub-range and captured the dissipative scales in the jet flow.

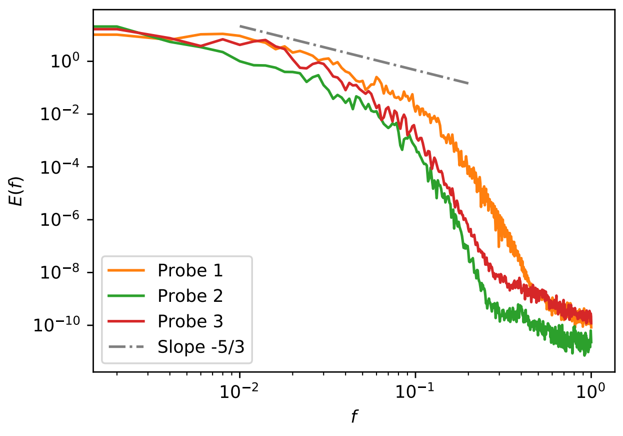
5.2 Running LES based on URANS data through a Screenshot method

The screenshot method is intending to use the URANS results at a specific time instant t_1 as initial conditions of a LES simulation to accelerate the reactor safety analyses. Compared with the LES, URANS needs less computational resources and time. The screenshot method can speed up the entire procedures. In the present study, a URANS result from a coarse mesh at $t_1 = 20.1\,s$ is adopted as the LES initial condition (see Figure 5.8a). The velocity and temperature fields are mapped by using the `mapFields` utility in OpenFOAM [31]. Two additional scenarios are added to study the velocity noise effects. The anisotropic velocity noise is from the result of $t = 20.1\,s$ at Simulation S5. 5% and 10% of velocity magnitude values are added to the URANS result, respectively.

The internal velocity fields at various time instants of simulations with different noises are shown



(a) Velocity magnitudes of various probes.



(b) Turbulence energy spectra of various probes.

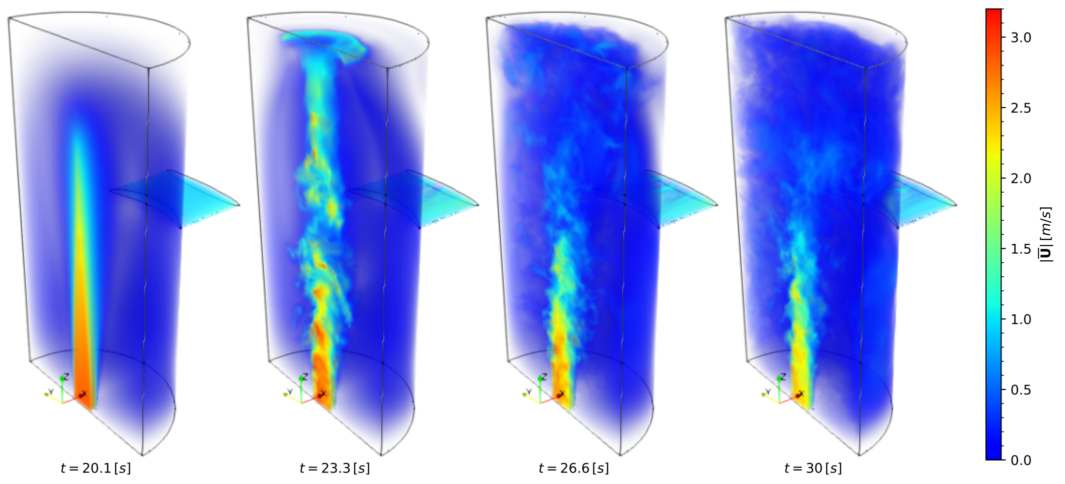
Figure 5.7: Velocity magnitudes and turbulence energy spectra of various probes in Simulation S5.

in Figure 5.8. The corresponding LES results are shown in Figure 5.9. It is clear shown that the results with different noises have no significant difference with each other. In the time of 23.3 s, the screenshot methods have high velocity regions close to the top surface of the tank, which are the propagation of the URANS solutions. Then, in the time of 26.6 s, the top region velocities decreased and became more close to the LES result. In the end, the velocity distributions from screenshot method near the potential core region and outlet patch are very close to the LES data, whereas the deviations are greater in the regions near the top surface.

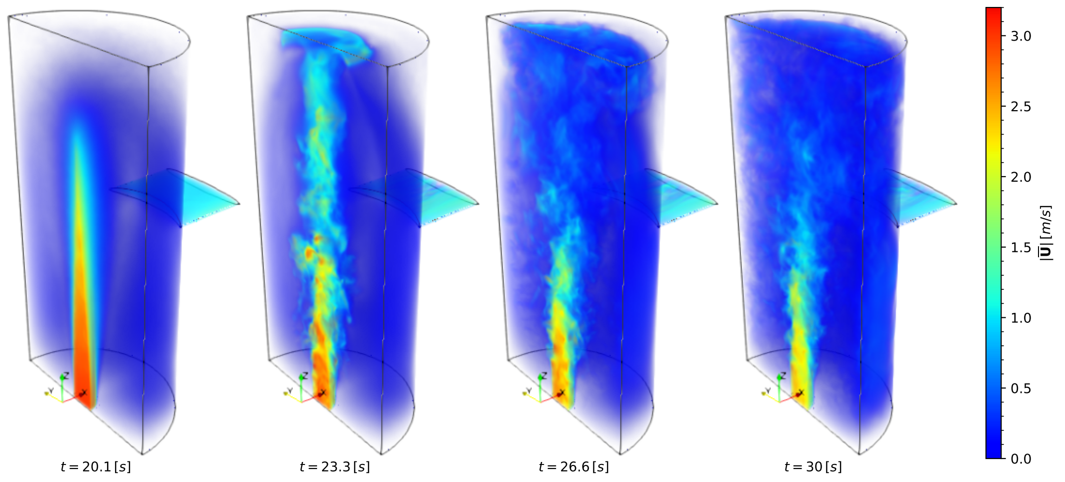
Figure 5.10 shows the time history of instantaneous velocity magnitudes at different probes. It is clear that the screenshot method results agree well with the LES one, and the added noises do not affect the velocity field very much in the condition of sufficient small convergence residuals in the outer iterations. It should be noted that the peak velocity values from screenshot method are smaller than the ones of the LES ones. At Probe 3, which is inside of the outlet channel, the velocity agreements between different methods are still acceptable. However, the results at Probe 2, which is near the top surface, deviates a lot. The velocity values from the screenshot method are relatively greater than the LES results.

The time history of average velocity magnitudes at different probes is shown in Figure 5.11. Same as the instantaneous velocity comparisons, the average velocity values are almost identical at Probe 1. Figure 5.11c also shows a good agreement between the screenshot method and LES results. In addition, the average velocity deviations are increasing at Probe 2.

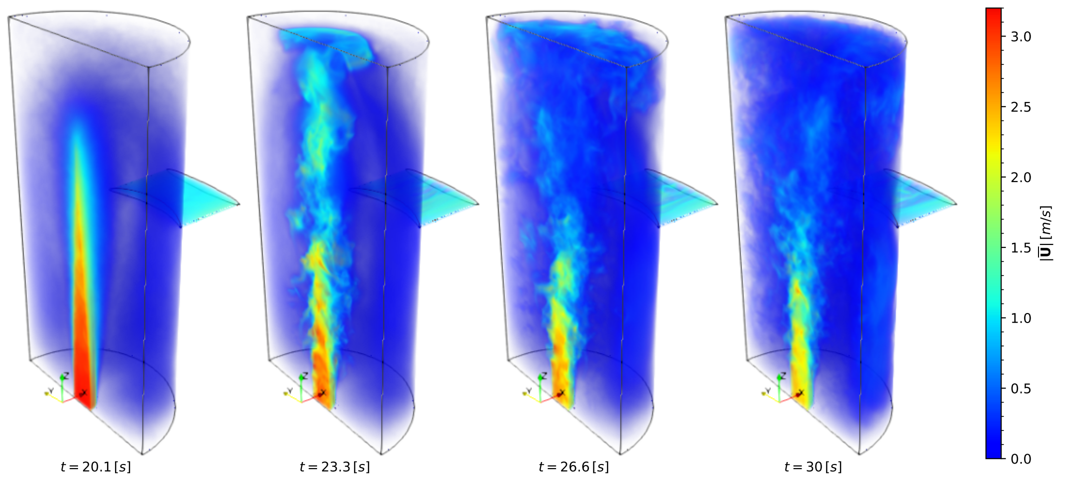
The results in the present study confirmed that the screenshot method could be employed to to accelerate the LES simulations and reactor safety analyses.



(a) Without noise



(b) 5% noise



(c) 10% noise

Figure 5.8: Internal velocity fields at various time instants of simulations with different noises.

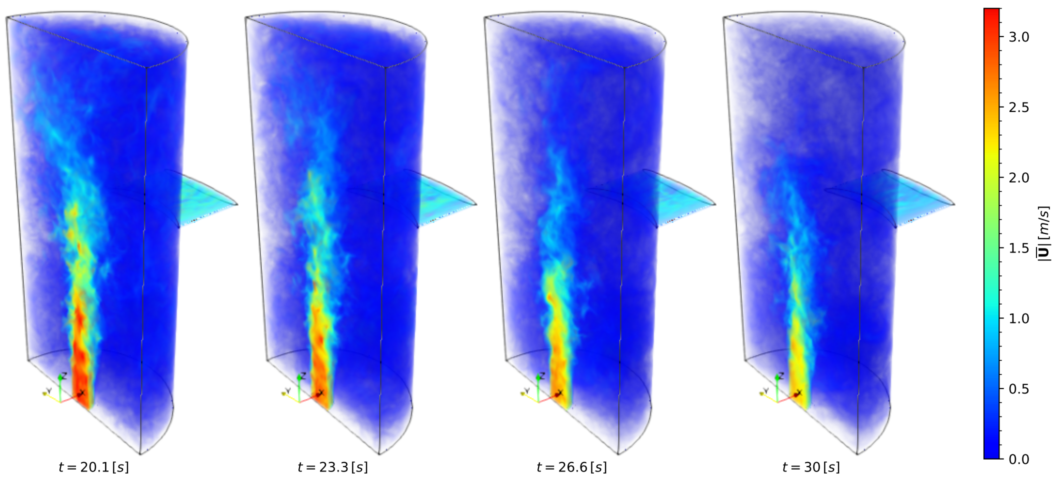
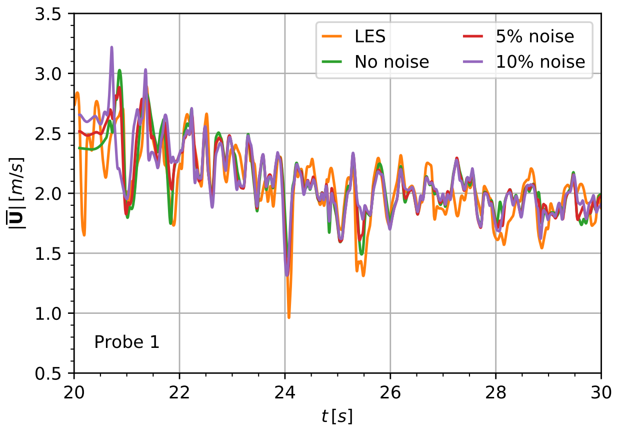
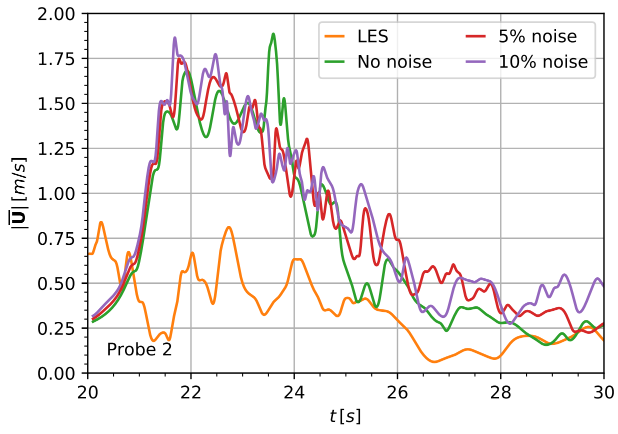


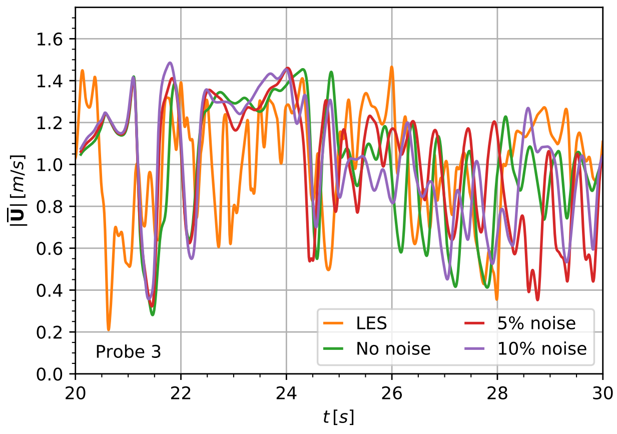
Figure 5.9: Internal velocity fields at various time instants of LES simulation.



(a) Probe 1.

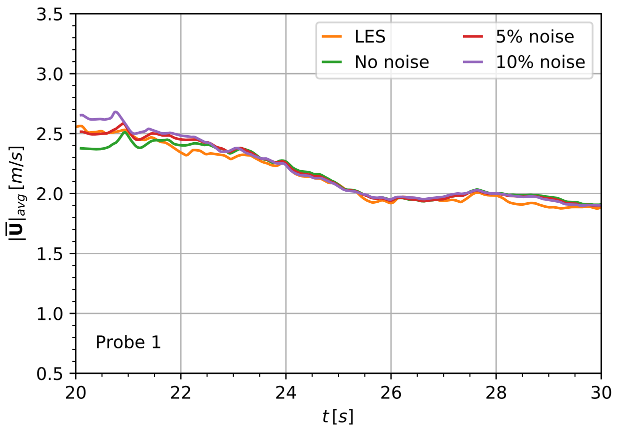


(b) Probe 2.

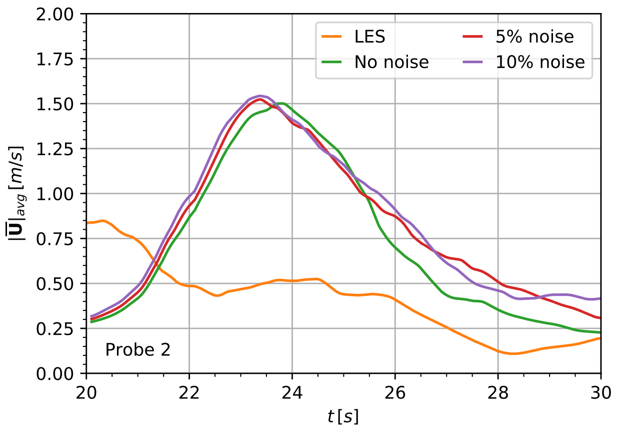


(c) Probe 3.

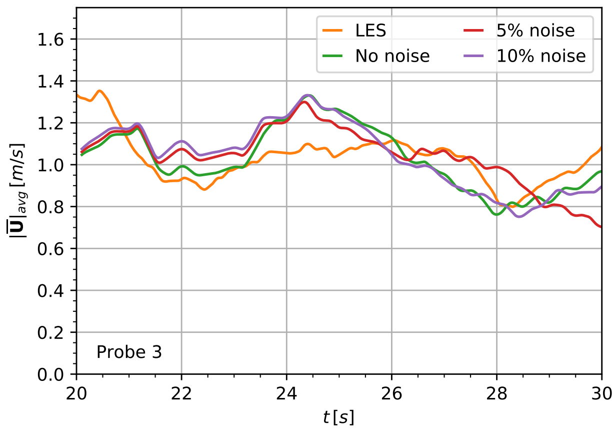
Figure 5.10: Instantaneous velocity magnitudes at different probes.



(a) Probe 1.



(b) Probe 2.



(c) Probe 3.

Figure 5.11: Average velocity magnitudes at different probes.

5.3 Improving ML-based closure through coupling LES and URANS data

With the support of the newly obtained high-fidelity LES data, we can further improve the ML-based closure by incorporating more physical domain knowledge. One possible approach is to expand the current DCNN-LSTM model into a composite neural network [36]: the previously developed DCNN-LSTM trained with RANS data will serve as a baseline model; another neural network will serve as a discrepancy correction term to the baseline DCNN-LSTM model. The new neural network will be trained with the newly available higher-fidelity LES data, which has a smaller dataset that covers only a few snapshots of the transient considering the computational expensiveness of LES. The new discrepancy correction network will incorporate physical domain knowledge by choosing a set of input features that are non-dimensional and consistent with the Galilean invariance under the isotropic flow assumption (i.e. translational invariance and uniform motion invariance) [37]. The two networks, combined together, will be fed into a physics-informed neural network [38] to ensure the prediction satisfies the structure of the governing partial differential equations. Such a proposed composite neural network structure is depicted in Figure 5.12.

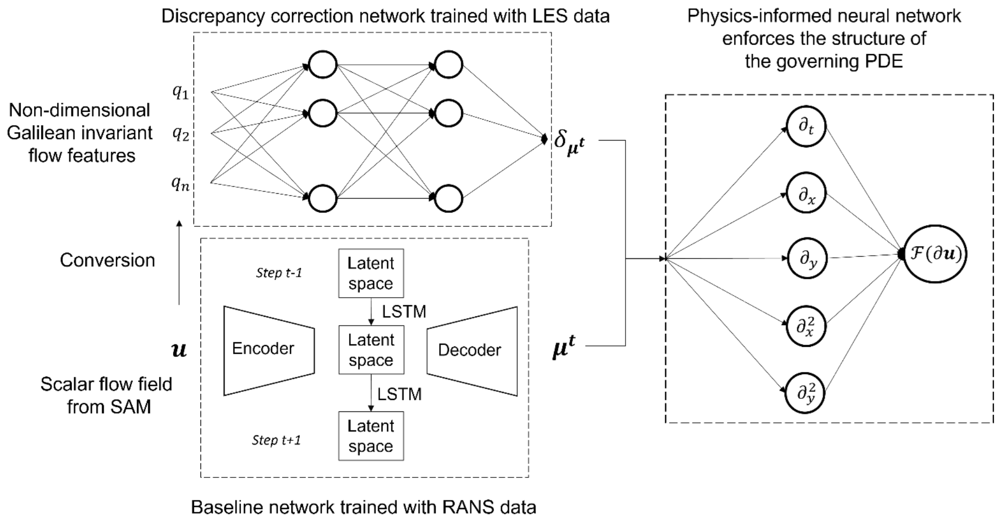

Discrepancy correction network trained with LES data
Physics-informed neural network enforces the structure of the governing PDE
Non-dimensional Galilean invariant flow features
q_1
q_2
q_n
δ_{μ^t}
∂_t
∂_x
∂_y
∂_x^2
∂_y^2
$\mathcal{F}(\partial \boldsymbol{u})$
Conversion
Step t-1
Latent space
LSTM
Encoder
Latent space
Decoder
$\boldsymbol{u}$
$\boldsymbol{\mu}^t$
Scalar flow field from SAM
LSTM
Step t+1
Latent space
Baseline network trained with RANS data

Figure 5.12: Composite neural network incorporates physical domain knowledge with the support of LES and URANS data.

6 Summary and future work

6.1 Summary

In this report, we proposed a data-driven modeling approach based on deep neural networks to develop ML-based closure to support coarse-mesh multi-dimensional modeling in modern nuclear-system code SAM. We demonstrated the applicability of this approach with a data-driven turbulence closure applied during loss-of-flow transients that involve thermal mixing and stratification phenomena in a pool-type sodium-cooled fast reactor. The proposed turbulence model has a coarse-mesh setup to ensure computational efficiency, while it is trained with fine-mesh CFD data to ensure accuracy. We developed, trained, and tested three neural network architectures for this purpose: the long-short-term-memory (LSTM) network based on proper orthogonal decomposition (POD), the densely connected convolutional neural network (DCNN), and the combination of DCNN and LSTM. We found that the DCNN-LSTM model can efficiently learn the spatial-temporal information from the CFD transient-simulation results. Utilizing the fine-mesh CFD simulation results from a loss-of-flow transient, we performed a comprehensive training and hyperparameter optimization process to obtain a DCNN-LSTM model with optimal performance. We then evaluated the generalization capability of the optimized DCNN-LSTM with two transients that have "extrapolating" inlet conditions compared to the training transient.

We found that the DCNN-LSTM model can predict the turbulent viscosity with reasonable accuracy over the whole training transient. This observation confirms the effectiveness of the training and optimization process. For the two cases with "extrapolating" inlet conditions, the DCNN-LSTM model could still capture the general distribution of the turbulent viscosity field, but showed discrepancies compared to the original CFD results. This observation indicates that the DCNN-LSTM model did have moderate generalization capability for the "extrapolating" inlet conditions. We then employed the t-distributed stochastic neighbor embedding algorithm to study the data similarity between the training transient and the two "extrapolating" transients. We found that the physical fields of the three transients have certain similarities, but the similarity is not great enough to fully cover the whole transients, so the model is less accurate on the transients with extrapolating inlet conditions compared to the training transient. Be that as it may, the results still provide promising insights indicating that we can leverage a well-trained data-driven model for simulation cases with "extrapolating" inlet and boundary conditions.

We further performed uncertainty quantification (UQ) for the ML model. In this first-of-the-kind work, the UQ is performed on a simplified 2D case on the DCNN model, but the work can be extended to 3D case without significant modification. We tested three UQ methods for ML model: Monte Carlo dropout, deep ensemble, and Bayesian neural network. The results demonstrate that deep ensemble and Bayesian neural network have good performance on capturing the uncertainty of the ML model, while the deep ensemble method showed better scalability and easier implementation compared to Bayesian neural network. We then developed the SAM-ML capability to enable efficient data exchange between SAM and the ML model, as well as involving ML model prediction in SAM's JFNK numerical solving process. A 2D case study on SAM-ML demonstrates that implementing the ML-based closure into SAM's multi-D module can improve the code's ability to capture turbulence effect during reactor transient while keeping its coarse mesh setup.

The framework developed in this report demonstrates the great potential of leveraging ML methods to assist the safety modeling of advanced reactors with both accuracy and computational efficiency. The developed framework can be applied to multiple of challenging problems in advanced reactor modeling and simulation that involve complex thermal-fluid phenomena. This proposed approach will be able to facilitate the development and deployment of advanced reactors by improving economics (through accurate safety margin predictions) and reducing the licensing burden (through improved uncertainty quantification).

6.2 Future work

We plan to further advance the investigation in two directions. The first is to further improve the performance of the ML-based closure, especially for "extrapolating" inlet/boundary conditions. The second is to develop additional applications based on the developed SAM-ML capability.

For the first task, two parallel approaches can be investigated. The first is to train the model with more transients that cover a variety of inlet/boundary conditions, so that the trained DCNN-LSTM model can generalize over a broad range of local flow features. In this way, for a case with "extrapolating" inlet/boundary conditions, the simulation results can find similarities with the training data more easily. The second approach is to embed physics directly into the data-driven model so the model can generate results that are consistent with the physical system. One example is physics-informed neural networks, which have enabled significant progress on many physical problems involving the solution of PDEs. Particularly, as discussed in 5.3, one can combine both approaches to improve the ML model. One example is the composite neural network [36], where multiple neural networks are trained with data of different fidelity, e.g. a small set of high fidelity DNS/LES data and a larger set of moderate-fidelity RANS data, these networks can be combined with embedded physics to achieve good generality.

For the second task, the developed SAM-ML capability that integrates ML with nuclear system code be leveraged for multiple purposes besides the ML-based closure. A few examples are: active learning for experiment design, online dimensionality reduction for compressed simulation results, simulation control supported by reinforcement learning, coupling numerical solver with ML in customized loss function construction.

Acknowledgments

This material is based upon work supported by Laboratory Directed Research and Development (LDRD) funding from Argonne National Laboratory, provided by the Director, Office of Science, of the U.S. Department of Energy under Contract No. DEAC02-06CH11357.

The authors gratefully acknowledge the computing resources provided on Blues, a high-performance computing cluster operated by the Laboratory Computing Resource Center at Argonne National Laboratory.

The authors gratefully acknowledge the resources of the Argonne Leadership Computing Facility, which is a DOE Office of Science User Facility supported under Contract DE-AC02-06CH11357.

References

[1] R. Hu, L. Zou, G. Hu, D. Nunez, T. Mui, and T. Fei, "SAM theory manual," Tech. Rep. ANL/NE-17/4 Rev. 1, Argonne National Lab.(ANL), Argonne, IL (United States), 2021.

[2] R. Hu, "Three-dimensional flow model development for thermal mixing and stratification modeling in reactor system transients analyses," *Nuclear Engineering and Design*, vol. 345, pp. 209–215, 2019.

[3] L. Zou, D. Nunez, and R. Hu, "Development and validation of sam multi-dimensional flow model for thermal mixing and stratification modeling," Tech. Rep. ANL-NSE-20/19, Argonne National Lab.(ANL), Argonne, IL (United States), 2020.

[4] K. Hornik, "Approximation capabilities of multilayer feedforward networks," *Neural networks*, vol. 4, no. 2, pp. 251–257, 1991.

[5] C. J. Permann *et al.*, "MOOSE: Enabling massively parallel multiphysics simulation," *SoftwareX*, vol. 11, p. 100430, 2020.

[6] Y. Chang *et al.*, "Advanced burner test reactor preconceptual design report.," Tech. Rep. ANL-ABR-1, Argonne National Lab.(ANL), Argonne, IL (United States), 2008.

[7] P. Siemens, "Simcenter star-ccm+ user guide.," tech. rep., Siemens PLM, 2018.

[8] A. Kraus, S. Aithal, A. Obabko, E. Merzari, A. Tomboulides, and P. Fischer, "Erosion of large-scale gaseous stratified layer by a turbulent jet—simulations with URANS and LES approaches," in *16th Int. Topl. Mtg, Nuclear Reactor Thermal Hydraulics (NURETH-16)*, American Nuclear Society, 2015.

[9] Y. Liu, N. Dinh, Y. Sato, and B. Niceno, "Data-driven modeling for boiling heat transfer: using deep neural networks and high-fidelity simulation results," *Applied Thermal Engineering*, vol. 144, pp. 305–320, 2018.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] Y. LeCun, Y. Bengio, *et al.*, "Convolutional networks for images, speech, and time series," *The handbook of brain theory and neural networks*, vol. 3361, no. 10, p. 1995, 1995.

[12] Y. Liu, R. Hu, P. Balaprakash, A. Brunett, and A. Obabko, "Coarse mesh turbulence prediction for reactor transient simulations using densely connected convolutional networks," in *2020 ANS Virtual Winter Meeting*, American Nuclear Society, 2020.

[13] B. Dong, Q. Jiang, and Z. Shen, "Image restoration: Wavelet frame shrinkage, nonlinear evolution pdes, and beyond," *Multiscale Modeling & Simulation*, vol. 15, no. 1, pp. 606–660, 2017.

[14] Z. Wang, K. Luo, D. Li, J. Tan, and J. Fan, "Investigations of data-driven closure for subgrid-scale stress in large-eddy simulation," *Physics of Fluids*, vol. 30, no. 12, p. 125101, 2018.

[15] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger, "Densely connected convolutional networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 4700–4708, 2017.

[16] Y. Zhu and N. Zabaras, "Bayesian deep convolutional encoder–decoder networks for surrogate modeling and uncertainty quantification," *Journal of Computational Physics*, vol. 366, pp. 415–447, 2018.

[17] Y. Gal and Z. Ghahramani, "Dropout as a bayesian approximation: Representing model uncertainty in deep learning," in *international conference on machine learning*, pp. 1050–1059, PMLR, 2016.

[18] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-C. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *Advances in neural information processing systems*, pp. 802–810, 2015.

[19] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *the Journal of machine Learning research*, vol. 12, pp. 2825–2830, 2011.

[20] P. Balaprakash, M. Salim, T. D. Uram, V. Vishwanath, and S. M. Wild, "Deephyper: Asynchronous hyperparameter search for deep neural networks," in *2018 IEEE 25th international conference on high performance computing (HiPC)*, pp. 42–51, IEEE, 2018.

[21] R. Maulik, A. Mohan, B. Lusch, S. Madireddy, P. Balaprakash, and D. Livescu, "Time-series learning of latent-space dynamics for reduced-order model closure," *Physica D: Nonlinear Phenomena*, vol. 405, p. 132368, 2020.

[22] L. Van der Maaten and G. Hinton, "Visualizing data using t-sne.," *Journal of machine learning research*, vol. 9, no. 11, 2008.

[23] X. Wu, T. Kozlowski, H. Meidani, and K. Shirvan, "Inverse uncertainty quantification using the modular bayesian approach based on gaussian process, part 1: Theory," *Nuclear Engineering and Design*, vol. 335, pp. 339–355, 2018.

[24] Y. Liu, N. T. Dinh, R. C. Smith, and X. Sun, "Uncertainty quantification of two-phase flow and boiling heat transfer simulations through a data-driven modular bayesian approach," *International Journal of Heat and Mass Transfer*, vol. 138, pp. 1096–1116, 2019.

[25] Y. Liu, X. Sun, and N. T. Dinh, “Validation and uncertainty quantification of multiphase-cfd solvers: A data-driven bayesian framework supported by high-resolution experiments,” *Nuclear Engineering and Design*, vol. 354, p. 110200, 2019.

[26] Y. Liu, D. Wang, X. Sun, N. Dinh, and R. Hu, “Uncertainty quantification for multiphase-cfd simulations of bubbly flows: a machine learning-based bayesian approach supported by high-resolution experiments,” *Reliability Engineering & System Safety*, vol. 212, p. 107636, 2021.

[27] Y. Liu, R. Hu, and P. Balaprakash, “Uncertainty quantification of deep neural network-based turbulence model for reactor transient analysis,” in *Verification and Validation Symposium 2021*, vol. 84782, p. V001T11A001, American Society of Mechanical Engineers, 2021.

[28] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: a simple way to prevent neural networks from overfitting,” *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[29] B. Lakshminarayanan, A. Pritzel, and C. Blundell, “Simple and scalable predictive uncertainty estimation using deep ensembles,” *Advances in Neural Information Processing Systems*, vol. 30, 2017.

[30] C. Blundell, J. Cornebise, K. Kavukcuoglu, and D. Wierstra, “Weight uncertainty in neural network,” in *International Conference on Machine Learning*, pp. 1613–1622, PMLR, 2015.

[31] OpenCFD, *OpenFOAM - The Open Source CFD Toolbox - User Guide.* OpenCFD Ltd., United Kingdom, v2012 ed., Dec. 2020.

[32] J. Smagorinsky, “General circulation experiments with the primitive equations: I. the basic experiment,” *Monthly Weather Review*, vol. 91, no. 3, pp. 99–164, 1963.

[33] A. Montorfano, F. Piscaglia, and G. Ferrari, “Inlet boundary conditions for incompressible les: A comparative study,” *Mathematical and Computer Modelling*, vol. 57, no. 7-8, pp. 1640–1647, 2013.

[34] N. S. Dhamankar, G. A. Blaisdell, and A. S. Lyrintzis, “Overview of turbulent inflow boundary conditions for large-eddy simulations,” *Aiaa Journal*, vol. 56, no. 4, pp. 1317–1334, 2018.

[35] R. I. Issa, “Solution of the implicitly discretised fluid flow equations by operator-splitting,” *Journal of computational physics*, vol. 62, no. 1, pp. 40–65, 1986.

[36] X. Meng and G. E. Karniadakis, “A composite neural network that learns from multi-fidelity data: Application to function approximation and inverse PDE problems,” *Journal of Computational Physics*, vol. 401, p. 109020, 2020.

[37] J. Ling, A. Kurzawski, and J. Templeton, “Reynolds averaged turbulence modelling using deep neural networks with embedded invariance,” *Journal of Fluid Mechanics*, vol. 807, pp. 155–166, 2016.

[38] M. Raissi, P. Perdikaris, and G. E. Karniadakis, “Physics-informed neural networks: A deep learning framework for solving forward and inverse problems involving nonlinear partial differential equations,” *Journal of Computational Physics*, vol. 378, pp. 686–707, 2019.